PILOT INVESTIGATION OF EFFECTIVENESS OF REAL-TIME FEEDBACK-ASSISTED PARTIAL WEIGHT BEARING

by

Alvin Yong-Been Le

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Bioengineering

The University of Utah

December 2015

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Alvin Yong-Been Le**

has been approved by the following supervisory committee members:

Tomasz Petelenz	, Chair	**6/15/2015**
		Date Approved
Robert Hitchcock	, Member	**6/15/2015**
		Date Approved
Kelly Broadhead	, Member	**6/12/2015**
		Date Approved

and by **Patrick Tresco**, Chair/Dean of

the Department/College/School of **Bioengineering**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

Partial weight bearing promotes bone formation during fracture rehabilitation and can result in a faster recovery period. However, prescribed partial weight bearing load levels are often exceeded, which can lead to disjunction. On the other hand, persistent under loading can delay healing. The effects of real-time feedback on a subject's ability to partially weight bear within a target load range is investigated in this experiment.

Ten healthy subjects recruited from the University of Utah were trained to partially weight bear 25% of their body weight on one leg. The subjects were shown how to perform 3-point gait on crutches and allowed time to become accustomed with walking on crutches. Subjects were instructed to partially weight bear within a target range of 20-30% of their body weight. Five subjects (Group 1) walked 50 steps without feedback, followed by 50 steps with real-time feedback. The other five subjects (Group 2) walked 50 steps with feedback, followed by 50 steps without feedback.

In both groups, the subjects' ability to partially weight bear improved when receiving real-time feedback. The average maximum loads with and without feedback were 24.87 ± 2.9 lbs. and 30.03 ± 11.5 lbs., respectively. The number of steps within the target load range of 20-30% were counted for each subject. A one-way ANOVA performed on the number of acceptable steps (within 20-30% body

weight) of the four test conditions (Group 1/Group 2 and Feedback/No feedback) found the F-ratio = 4.54 and F-crit(3,16) = 2.46, resulting in the rejection of the null hypothesis that all four population means were equal. A paired t-test performed on Group 1, pairing the number of acceptable steps without feedback with the number of acceptable steps with feedback, showed a significant statistical difference between the steps with and without feedback (P=0.0165). This indicates feedback significantly affected the subject's ability to partially weight bear within the target load range. A paired t-test was also performed on Group 2; the test showed no significant statistical difference (P=0.0958). This indicates learning occurred during the 50 steps when feedback was present, improving the subject's ability to weight bear without feedback for the next 50 steps.

Future work include utilizing the experimental data to power a larger study. Also, many methods of providing real-time feedback to patients are possible and the efficacy of each should be explored. For this study, visual representation of the current load was used as the method of providing feedback. However, other forms of feedback may lead to better patient outcomes and compliance.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

Thank you to my advisors and colleagues whom I have befriended during my time at the University of Utah. Thanks to those who have extended their generosity towards the development of my master's work.

Special thanks to my family, Chin, Bee, and Cynthia, for all of your support.

CHAPTER 1

PARTIAL WEIGHT BEARING AND REAL-TIME FEEDBACK

Partial Weight Bearing

When a limb is immobilized for long periods of time after surgery, the rate of bone loss begins to exceed the rate of bone formation when no load or a decreased amount of load is exerted on the limb.[1] This leads to loss of bone density and also decreases the strength of the bone.[1] Research in orthopedic practice has shown controlled cyclic loading helps accelerate the healing of bone and also counteracts bone density loss.[1],[2],[3] This controlled, cyclic loading can be accomplished by limiting limb loading to a fraction of a patient's full body weight – a practice called Partial Weight Bearing (PWB). Patients are instructed to limit load bearing on the healing limb to a fraction of their body weight, as prescribed by a physician. Commonly prescribed PWB values include 25%, 50%, and 75% of the patient's body weight. The patients are prescribed lower PWB values initially and value is increased as rehabilitation progresses.

The theory behind the practice of partial weight bearing is there are beneficial effects of controlling limb load exertion during a recovery period. This is based off of Wolff's Law which states: "bone in a healthy person or animal will adapt to the loads under which it is placed.[4] If loading on a particular bone increases, the bone will remodel itself over time to become stronger to resist that

sort of loading."[5],[6] In clinical practice today, patients are prescribed an upper and lower limb load limit by their medical care provider.[5] Setting a lower limit ensures there will be exertion of force on the limb, resulting in increased osteocyte activity and bone formation. An upper limit of limb loading limits the patient from exerting an excessive load on the recovering limb, preventing the risk of a repeat injury.

The benefits of partial weight bearing is supported by a few studies. One rat model demonstrated that controlled loading on a healing limb promotes callus formation. This helps accelerate the bone remodeling process.[7] A paper written by Meadows et al. 1990 examined the effect of weight bearing on defects in canine tibia. The canine model has previously been used for gait analysis. This study utilized three groups of canines, two experimental groups and one control group studied over a 28-day experimental period.[8],[9] Canines in the experimental groups were sedated and bilateral defects were made in the hind legs of the dogs. No tibial defects were made to the canines in the control group. Canines in experimental group 1 had one hind leg suspended in air to hinder weight-bearing on that limb. Canines in experimental group 2 were allowed to weight bear on all four limbs. The canines in the control group were allowed to weight bear on all four limbs. In Group 1, the limb suspended in air showed a decrease in formation of woven bone compared to the weight bearing limb with the identical tibial defect. A comparison of canines in Group 1 and Group 2 showed percentages of newly woven bone on the weight bearing limb with the tibial defect were nearly identical.

A problem with current PWB practice is patients are prone to exceed prescribed PWB values, which can result in nonunion of the fractured bone.[10] This

is primarily due to the lack of patient ability of quantifying limb load exertion. The current standard of care in prescribing a PWB regimen is to have patients perform a calibration by exerting force on a weight scale and memorizing the force they are exerting on the injured limb. Patients are expected match the load during ambulation with crutches, but without the benefit of feedback from a weight scale. In a study by Vasarhelyi et al. 2004, researchers explored the ability of patients to partially weight bear without feedback. Twenty-three patients (single limb isolated fracture) and eleven healthy volunteers were enrolled into the study. The subjects were instructed to partially weight bear using crutches while walking around using the 3-point gait method. The researchers prescribed a maximum force of 200N and the subject's perception of limb loading was calibrated by exerting 200N on a weight scale and receiving feedback from looking at the scale. The subjects were observed over a three-day period while wearing a device measuring force exerted during each step. The researchers found that all subjects exerted a force higher than 200N on average over the testing period.

Over the past few decades, researchers have been studying the possibility of using feedback to improve a subject's ability to partially weight bear within an acceptable load range.[11],[12],[13] In a study done by Hustedt et al. 2012, researchers trained patients to partially weight bear using different methods of intervention.[11] The three methods of intervention were: verbal instruction, training with a weight scale, and biofeedback training. Verbal instruction was performed by explaining the weight bearing goals to the subject. The weight scale training utilized a bathroom scale to allow patients to partially weight bear on the scale in order

calibrate their load bearing perception of the target load bearing value. The biofeedback training consisted of using a device that output audio cues to the patient when the upper or lower load bearing thresholds were crossed. The audio cues were only active during training and were turned off during the experimental procedure. After receiving the interventional training, the subjects in each group were instructed to weight bear for 50 steps at 75 lbs. The researchers found subjects given verbal instructions performed the worst with an average maximum load of 92.28 ± 7.85lbs. The subjects trained using the weight scale had an average maximum load of 90.82 ± 7.19 lbs. Subjects trained using feedback performed best with an average maximum load of 69.67 ± 3.18 lbs. A study done by Winstein et al. 1996 examined the effectiveness of real-time feedback compared to postresponse feedback.[14] Subjects were asked to partially weight bear 30% of their body weight while receiving postresponse feedback or real-time feedback and the load bearing values were recorded. The subjects were asked to return two days later to perform the same procedure, except this time without any feedback. The researchers found that real-time feedback was more beneficial for immediate performance and postresponse feedback had a higher impact on long-term learning.

ATLAS – Ambulatory Tibial Load Analysis System

In order to measure limb load exertion, an Ambulatory Tibial Load Analysis System (ATLAS) was developed. The ATLAS system provides medical practitioners with the ability to collect and analyze limb loading data in patients with lower extremity fractures, specifically with tibial fractures, during the rehabilitation

period. ATLAS uses a Don Joy Orthopedics Max Trax CAM Walker, which is equipped with three pressure sensors, as shown in Figures 1-1 and 1-2. The pressure sensors are embedded in a foam pad which are positioned under the heel, medial, and lateral portions of the foot, as shown in Figure 1-2. Attached on the side of the boot is a wireless-enabled data recorder that samples load data and transmits in real-time to a mobile device for display, in this case a tablet computer.

The display module uses an Android operating system based application for receiving limb load data from the ATLAS-equipped CAMWalker. The application on the Android device serves as a load meter that displays the current total load exerted on the limb to the user, as seen in Figure 1-3. The load meter changes between gray, green, or red depending on if the current total load is below, within, or above the specified target range, respectively. The bar graph shows the maximum total force of the five previous steps. This display was used to provide subjects with real-time feedback. For this study, the acceptable green region was 20%-30% of the subject's body weight with 25% body weight being the target load value. The range of ± 5% was chosen due to the latency of subject response to real-time biofeedback.[15] The latency is attributed to the period of time between auditory or visual perception and motor response.

Figure 1-1. A side view of ATLAS with electronics module

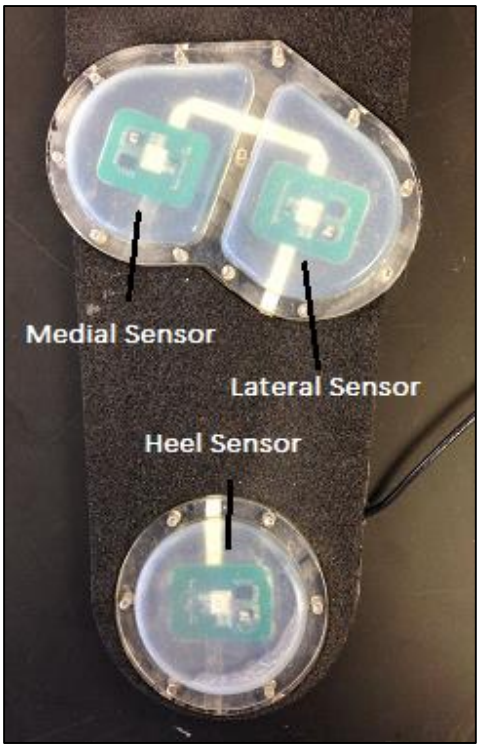

Figure 1-2. ATLAS pressure sensor configuration

.

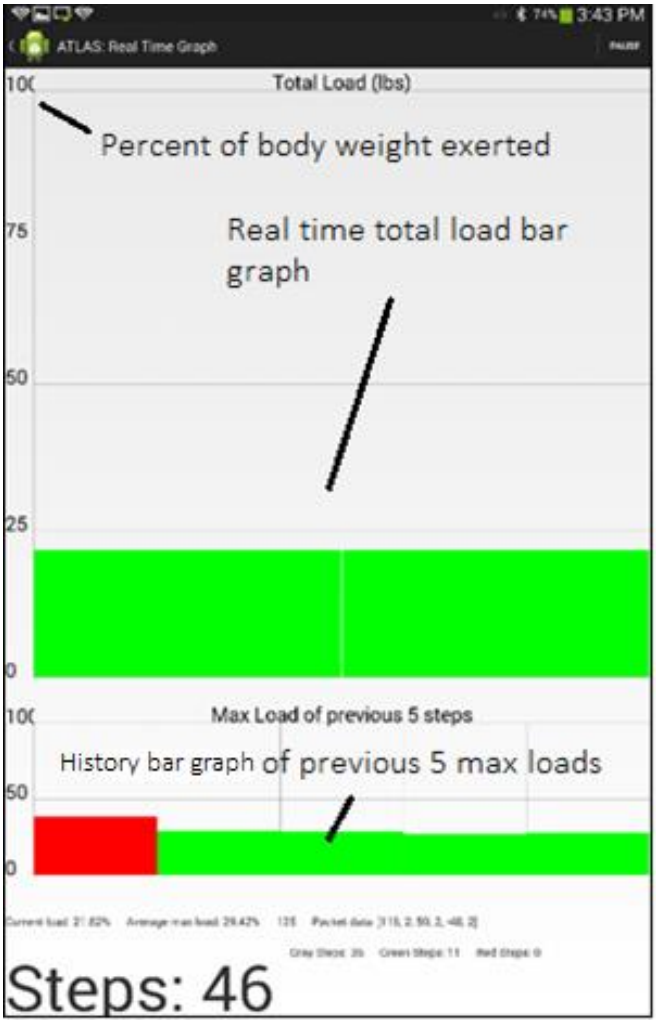

Figure 1-3. ATLAS app real-time load meter (top) and 5 step max load history bar graph (bottom)

CHAPTER 2

OBJECTIVE, HYPOTHESIS, AND STUDY DESIGN

Objective

The objective is to test the effects of real-time feedback on a subject's ability to partially weight bear within 20% to 30% of their body weight.

Hypothesis

Subjects will be able to partially weight bear within 20% to 30% of their body weight more consistently when they are receiving real-time feedback from the Atlas App.

Design

This study was designed as a minimal risk pilot study of real-time feedback-assisted PWB ambulation in healthy human volunteers. University of Utah students with no lower limb injuries affecting regular ambulation were selected to participate in this study. Subject were asked to partially weight bear with and without feedback in order to study the effect real-time feedback has on a subject's ability to partially weight bear within the target load range of 20%-30% of the subject's body weight. This study was submitted to the University of Utah Institutional Review Board and was determined to be a nonsignificant risk study . Due to this, it was exempt from further IRB review under Exemption Category 10 on 2/10/2015.

The IRB reference number assigned to the study is IRB_00079766.

Participant Selection Criteria

Inclusion Criteria

- 18-70 years old
- No lower extremity injuries affecting normal walking

Exclusion Criteria

- Less than 18 years old.
- Older than 70 years old.
- Individuals currently dealing with any type of injury that affects normal walking

Study Procedures

Recruitment/ Participant Identification Process

The research team recruited University of Utah students, faculty, and staff. Recruitment was done by informing participants of this study in person, placing informational flyers around the University of Utah Campus, and also calling known associates.

Consent Process

All prospective participants received detailed information about the study from research personnel trained in the protocol before they gave consent at the University of Utah. If the subject was interested in participating, the subjects considered an IRB-approved human subject information consent form, which they

signed if they chose to participate. The subjects were able to ask the researchers any questions they had about the study.

Procedures

After consent was received, the subject was asked to stand on a scale in order to obtain their body weight. The subject's body weight was used to calculate the acceptable load range of 20%-30% body weight in pounds.

The subject was then asked to place and secure their foot into ATLAS. Partial weight bearing requires the subject to walk with crutches, so subjects were trained to walk with crutches using 3-point gait. After being trained, a pair of red-dot adult Guardian crutches were adjusted to fit the subject. The subject was allowed to practice walking with 3-point gait until they signaled that they were comfortable. Half of the subjects performed the control procedure followed by the experimental procedure and the other half performed the experimental procedure followed by the control procedure. The order of the procedure each subject would perform was randomized prior to enrollment.

Control Procedure

While wearing ATLAS, the subject was asked to exert 25% of their body weight on the ATLAS wearing foot onto a weight scale while using crutches to PWB. Once the subject could consistently PWB 25% of their body weight for several seconds, the subject was asked to memorize how to PWB at 25% body weight. The subject was then asked to walk 50 steps on even terrain with crutches using 3-point gait while trying to replicate PWB 25% during each step. At no time

during the control procedure did the subject receive any feedback from the Atlas app. The researchers were able to see the current load on the app, but at no point did the researchers give any indication of performance to the subjects during the control procedure.

Experimental Procedure

While wearing ATLAS, the subject was asked to exert 25% of their body weight on the ATLAS wearing foot onto a weight scale while using crutches to PWB. Once the subject was consistently able to PWB 25% of their body weight for several seconds, the subject was asked to memorize how to PWB at 25% of their body weight.

The subject was then asked to walk 50 steps on even terrain with crutches using 3-point gait while trying to weight bear 25% of their body weight while receiving real-time feedback from the Atlas app. A researcher walked in front of the subject holding the Android device with the Atlas app launched on the screen. At no time during the walking phase of the experimental procedure did the researcher view the display of the Atlas app or attempt to influence the performance of the subject.

Statistical Analysis

A paired t-test was used to determine if there was a statistically significant difference between the 50 steps with feedback and the 50 steps without feedback. A one-way ANOVA was performed on the number of acceptable steps of the four test condition populations: Group 1 no feedback, Group 1 feedback, Group 2 no

feedback, and Group 2 feedback. Two paired t-tests were completed, one for the five subjects performing the control procedure first and one for the five subjects performing the experimental procedure first. The average max loads and standard deviations for both groups were also calculated.

CHAPTER 3

MSP430F5529 MICROCONTROLLER

Overview

This section discusses the electronic hardware of Atlas and the Low Energy Bluetooth protocol for transmission of data to an Android device. The Atlas module is controlled by a Texas Instrument MSP430F5529 Microcontroller. The essential functions of the microcontroller include data acquisition timing, communication, and data transfer between the sensor array and the Bluetooth module, Figure 3-1.

In the flow diagram shown, data are sampled every 62.5ms from the sensor array. The load values are stored within the microcontroller prior to being transferred into a buffer for transmission to the low energy Bluetooth module using .a Universal Asynchronous Receiver Transmitter (UART), an integrated circuit which translates data between parallel and series forms.

Load Data Sampling

The Atlas sensor array consists of three sensors (heel, medial, lateral), each having 14-bit resolution to represent a limb load range of 0 to 250 pounds. The transmission buffer of the microcontroller is an 8-bit buffer. So in order to transmit the data, each 14-bit measurement was split into two 8-bit variables prior to transmission and reassembled into a 14-bit variable after the data were received by the Android app. The first 8-bits of the heel measurement are assigned to the

uint8 variable h1 and the last 6-bits are assigned to the uint8 variable h2 using a-bit shift operation. The same process is done for the medial and lateral 14-bit measurements, shown in Figure 3-2.

After each of the 14-bit measurements are broken down into two 8-bit variables, they are sent to the Bluetooth module via UART communication. The order of the packet is: h1,h2,m1,m2,l1,l2, where h represents the heel load value, m represents the medial load value, and l represents the lateral load value. The order in which the data are sent from the MSP430F5529 microcontroller to the BLE113 Low Energy Bluetooth module is shown in Figure 3-3.

Timing of Data Sampling and Communication

An oscilloscope was used to validate the timing of sampling rate and data transmission. The key parameters were the integrity of data transmission as well as flow control timing in relation to each sample recorded by the microcontroller. In order to keep synchronized timing between the microcontroller and the Bluetooth module, a one-way flow control system was implemented. Prior to the transmission to each packet of data, a digital input/output pin of the microcontroller was set to high for a brief period of time before switching to the low state. This was used to signal to the Bluetooth module that the microcontroller had sampled new data and was ready to transmit to the Bluetooth module using UART communication.

The purple trace in Figure 3-4 shows a dip in supply voltage to the sensor array that occurs when the microcontroller samples data from the sensor array. The sampling rate of the microcontroller is set at 16Hz or 62.5ms between each

sample. In the image, the time between the purple vertical cursors is 62.14ms, which correlates with the expected sampling rate. The blue trace represents the UART communication between the microcontroller and the Bluetooth module.

The light blue trace in Figure 3-5 shows the data transmission to the BLE113 module using UART format. The vertical markers were placed at the beginning of the two data packets shown. The markers show a timing separation of 62.37ms, which is consistent with the 16Hz sampling rate of the sensor array. The spacing between the data transmissions (blue trace) from the microcontroller to the Bluetooth module shows that each set of sampled data are being transmitted to the Bluetooth module at a rate of 16Hz.

The timing between the microcontroller and the Bluetooth module is controlled using one-way flow control from the microcontroller. A digital I/O pin on the microcontroller is set to high briefly and immediately switched back to low. This signals to the Bluetooth module that there are data ready to be sent. The flow control signal is shown by the green trace in Figure 3-6. The timing distance between the flow control signal and the data being transmitted are less than 0.1ms.

The data being transferred from the microcontroller to the Bluetooth module use a Universal Asynchronous Receiver and Transmitter or UART. Each of the sensors have a 14-bit resolution between 0 and 250 pounds. The transmission buffer of the microcontroller is an 8-bit buffer, so data are broken down into two 8-bit variables using a-bit-shift operation. These data are recombined into a 14-bit integer after the data are received by the Android app. In Figure 3-7, six separate bytes of data can be seen separated by start and stop-bits. This figure shows all

of the sensors reading at a value of 0 while there is no load applied to the sensors. The second byte from the right-hand side has the second-bit showing as being a 1 due to noise on the sensor.

Figure 3-8 shows the wiring between the microcontroller and the BLE113 module. There are four connections between the MSP430F5529 and the BLE113 module. A 3.3V supplied by the voltage regulator on the circuit board is used to power the BLE113 module. The ground of the BLE113 is connected to the ground of the board. Pin 3.4 of the MSP430, the pin configured to output the flow control signal, is connected to pin 0.0 of the BLE113. Pin 3.3 of the MSP430, the pin which is configured to output the UART signal, is connected to pin 0.5 of the BLE113 module.

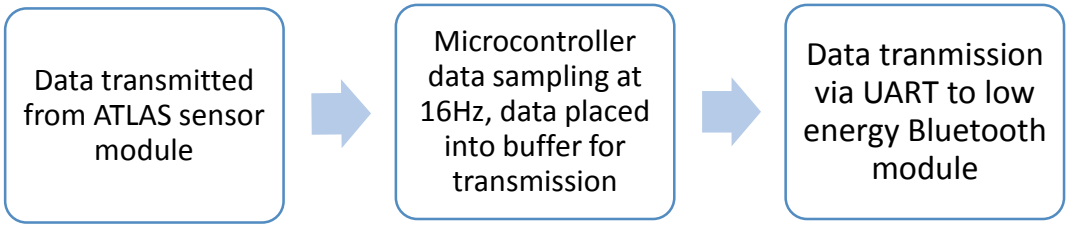

Figure 3-1. High level system flow chart for Atlas Microcontroller code. Data are sampled from the ATLAS sensor module, placed into a buffer, and transmitted to the Bluetooth module

```
void
send_uart(volatile sample_t *sample)
{
    P3OUT ^= BIT4;  // Toggle Flow Control

    P3OUT ^= BIT4;

    //Heel Data
    uint16_t heel16 = sample->heelCounts;
    uint8_t h1 = heel16;
    uint8_t h2 = (heel16>>8);

    //Medial Data
    uint16_t medial16 = sample->medialCounts;
    uint8_t m1 = medial16;
    uint8_t m2 = (medial16>>8);

    //Lateral Data
    uint16_t lateral16 = sample->lateralCounts;
    uint8_t l1 = lateral16;
    uint8_t l2 = (lateral16 >> 8);
```

Figure 3-2. A bit shift operation turning the 14-bit variable into two 8-bit variables

```
while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
UCA0TXBUF = h1;

while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
UCA0TXBUF = h2;

while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
UCA0TXBUF = m1;

while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
UCA0TXBUF = m2;

while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
UCA0TXBUF = l1;

while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
UCA0TXBUF = l2;
```

Figure 3-3. Transmission of 8-bit variables from the microcontroller to the Bluetooth module

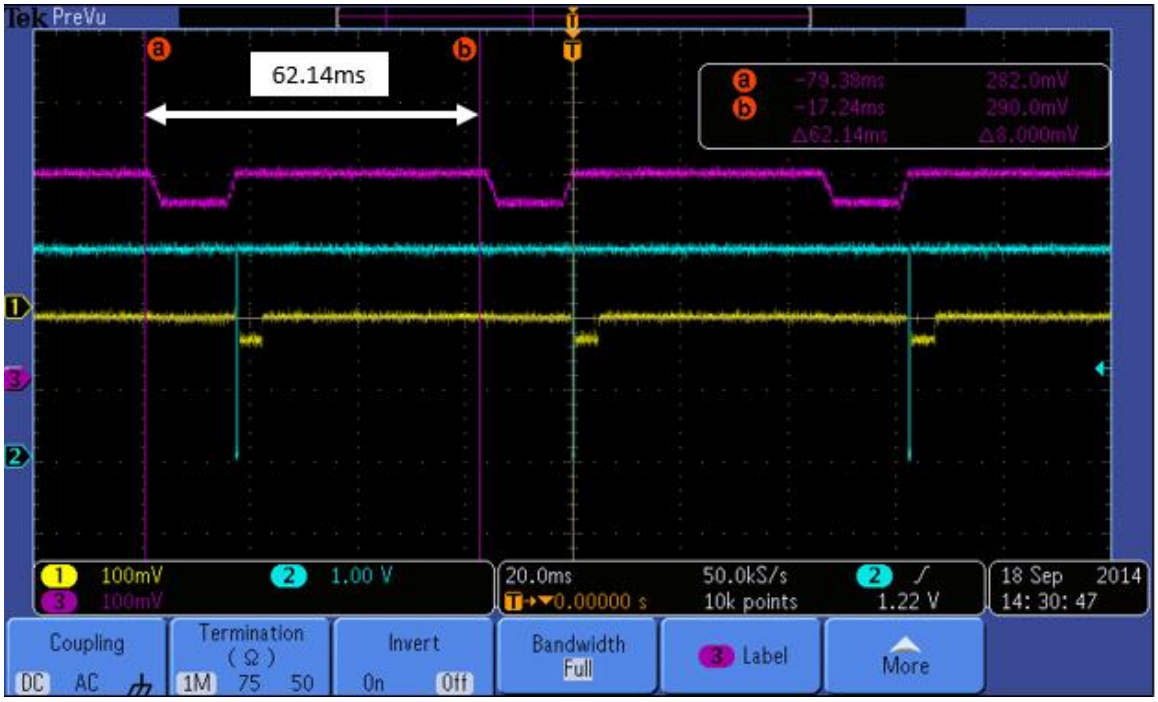

Figure 3-4. Timing of data sampling from ATLAS sensor array by the microcontroller. The data packets are shown to be 62.37ms apart, approximately 16Hz.

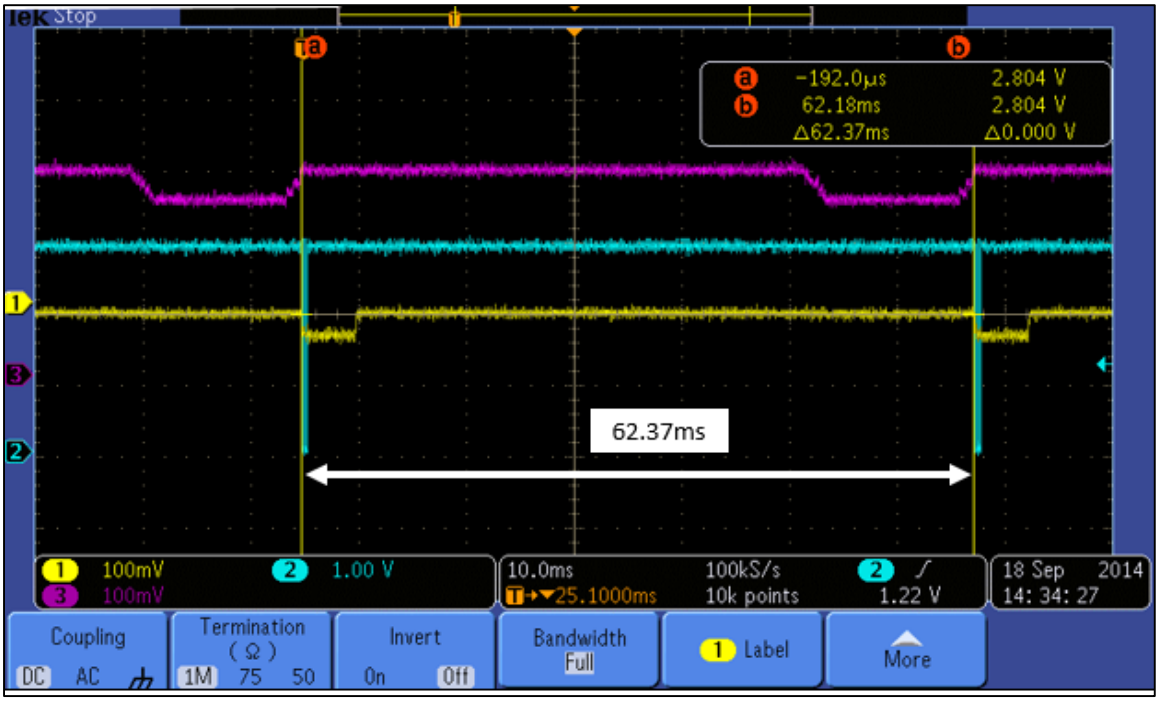

Figure 3-5. Timing of data transmission from microcontroller to BLE module. The data packets are shown to be 62.37ms apart, approximately 16Hz.

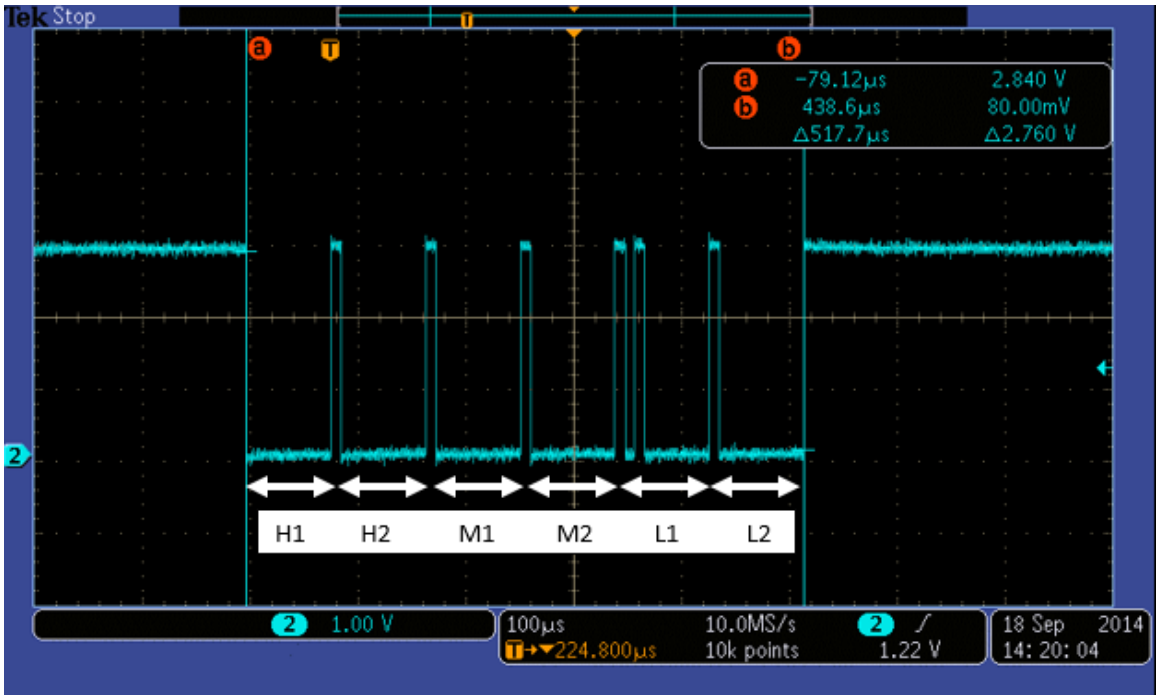

Figure 3-6. Timing of flow control and data transmission to BLE module. The timing between the flow control signal and data transmitted to the BLE module is 94.7us

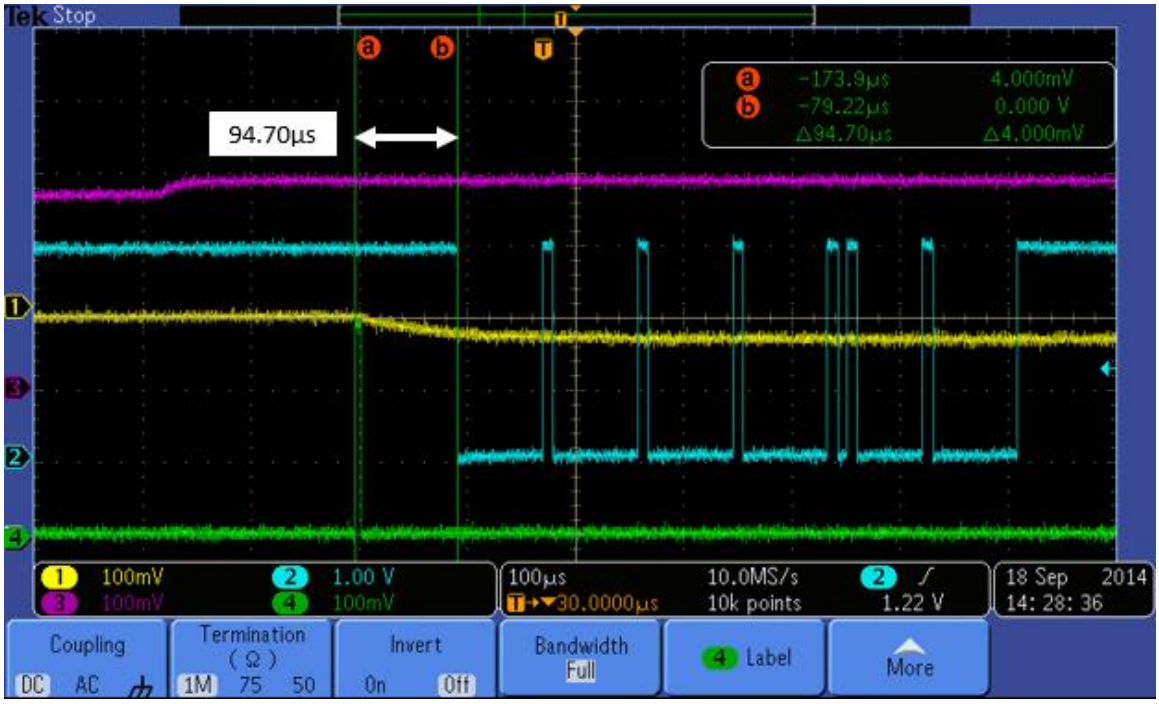

Figure 3-7. Timing of flow control and data transmission to BLE module. This data packet shows the six bytes of data when no load is present. The image shows the order of the bytes representing the 14-bit heel, medial, and lateral measurements. Where H stand for heel, M stand for medial and L stands for lateral.

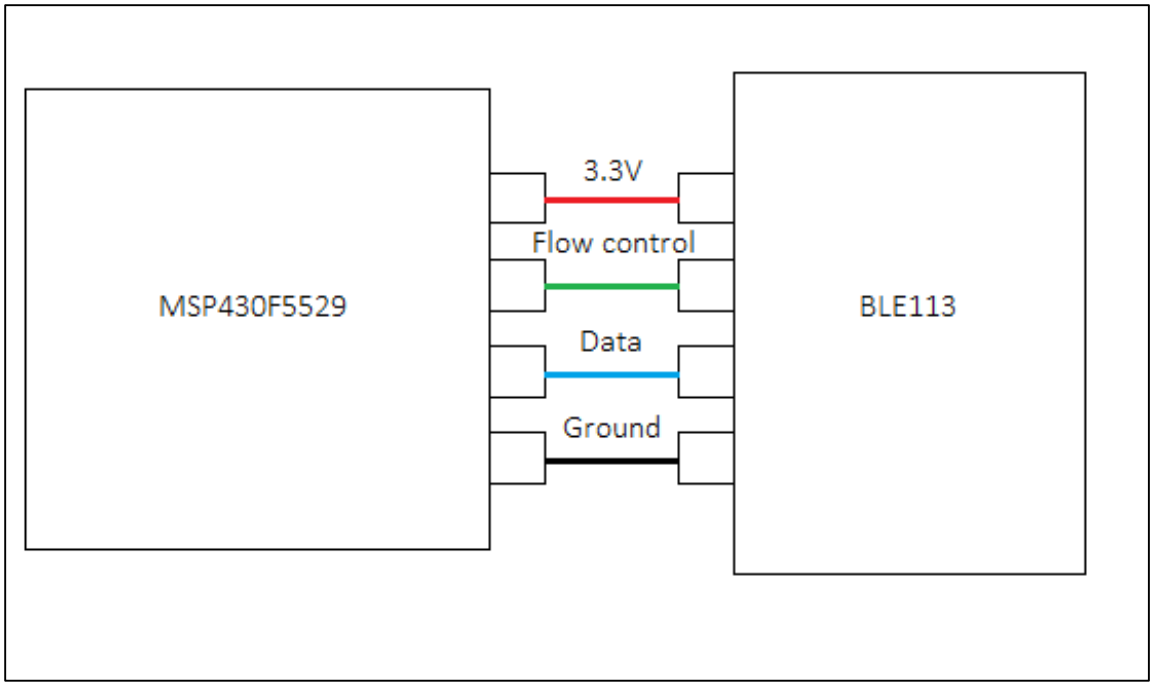

Figure 3-8. Schematic diagram of connections between MSP430F5529 and BLE113 module. The MSP430 P3.3 connects to BLE P0.5 for the data connection. The MSP430 P3.4 connects to BLE connects to BLE P0.0 for the flow control connection.

CHAPTER 4

BLE113 BLUETOOTH LOW ENERGY MODULE

Overview

A Bluegiga BLE113 Low Energy Bluetooth module was used to transfer data from the microcontroller to the Android device wirelessly. The BLE113 features a Bluetooth 4.0 low energy radio, an integrated antennae, and low power consumption during operation. The section below describes the interaction between the MSP430F5529 microcontroller and the BLE113 in regards to one-way flow control and data transmission.

Data Reception and Transmission

When the Bluetooth Module is initializing, the modules sets an interrupt on port P0.0 as shown in Figure 4-1. This pin is connected to the flow control pin of the microcontroller. One-way flow control is used to maintain synchronous timing between the microcontroller and the Bluetooth module and it ensures accurate data transmission between the two modules.

In the case Port P0.0 is pulled high, the Bluetooth module enters the interrupt code shown in Figure 4-2. The RX (receive) watermarks are set to enable the Bluetooth module to receive incoming data on the RX pin. Upon receiving data on the RX Pin, the system_endpoint_watermark_rx event is triggered as shown in

Figure 4-3. The number of bytes in the incoming data packet is passed into this event, as well as the endpoint which the data are received. If the endpoint matches the expected port, the length of the packet to be transmitted is set to the total number of bytes received. The maximum number of bytes that can be transmitted at one time is 20. If the data packet is larger than 20 bytes, all bytes past the 20th byte is not transmitted. The data are then written into an attribute in the GATT profile and broadcasted to the Android device.

```
event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)
    call system_endpoint_set_watermarks(5, 0, 0) # disable watermarks
    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)

    # enable interrupt on P0_0 and P0_1 rising edge
    # (parameters are port=0, bitmask=0b00000011, edge=rising)
    call hardware_io_port_config_irq(0, 3, 0)

end
```

Figure 4-1. Initialization and set-up of the BLE113 module occurs when event system_boot is called

```
event hardware_io_port_status(delta, port, irq, state)
    if port = 0 then
        if (irq & 1) = 1 then
            # P0_0 is HIGH and the source of this interrupt
            # DEVKIT UART OUTPUT: P0_0 pressed
            call system_endpoint_set_watermarks(5, 1, $ff) # set RX watermark
            call system_endpoint_set_watermarks(5, 0, $ff) # set RX watermark
        end if
    end if
end
```

Figure 4-2. Interrupt code for the BLE113 module. The interrupt pin is located on Port P0.0

```
event system_endpoint_watermark_rx(endpoint, size)

    if endpoint = 5 then
        in_len = size
        if in_len > 20 then
            in_len = 20
        end if


        call system_endpoint_rx(5, in_len)(result, in_len, in(0:in_len))
        call attributes_write(xgatt_data, 0, in_len, in(0:in_len))

    end if
end
```

Figure 4-3. Broadcast code for the BLE113 module. When attributes_write is called, the data received by the BLE113 module are broadcasted.

CHAPTER 5

ATLAS ANDROID APP

Overview

The purpose of the Atlas Android app is to display load data in real-time to a subject wearing the Atlas boot. This is accomplished by configuring the app to receive and process the data being broadcasted by the BLE113 low energy Bluetooth module. For this application, the operating system of the Android device has a requirement of running 4.3 Jellybean or higher due to the addition of Low Energy Bluetooth capabilities in OS 4.3 Jellybean. The data transfer using Bluetooth and data processing will be discussed first, followed by discussion of the graphical representation of the data. A high level flow system flow chart of the Atlas Android app is shown below in Figure 5.1.

Data Reception and Processing

When the Atlas app is first launched, it prompts the subject for their body weight in pounds as shown in Figure 5-2. The subject can enter their body weight and scan for available Bluetooth devices by hitting the “Find Devices” button. Pressing this button starts the MainActivity.java activity as well as passes the weight of the subject into the new activity. The MainActivity.java activity scans for nearby low energy Bluetooth devices and populates a list of selectable items with

the available devices in the vicinity, shown in Figure 5-3. In the case that no devices are available or a second scan of available devices is necessary, this list can be refreshed by pressing the scan button in the menu options. Pressing the scan button executes a command in the code to clear the previous list of available devices and repopulate a new list by scanning for available devices in the area, shown in Figure 5-4. An example of the generated list of available BLE113 devices is shown below in Figure 5-3. Upon clicking on one of these devices, the device name and address as well as the subject's body weight is passed onto a java activity called BarGraphing.java using the onListItemClick method and the BarGraphing.java activity is initialized, shown in Figure 5-5.

Real-time data transfer and updates are accomplished by using a BroadcastReceiver in the BarGraph.java activity, shown in Figure 5-6. Upon receiving any changes in the low energy Bluetooth connection, the intent is checked using the BroadcastReceiver. There are four possibilities of intents that can be passed to the BroadcastReceiver. ACTION_GATT_CONNECTED occurs when the device establishes a connection with a nearby Bluetooth Low Energy GATT server, ACTION_GATT_DISCONNECTED occurs when the device is disconnected from a Bluetooth Low Energy device or Bluetooth Low Energy GATT server, ACTION_GATT_SERVICES_DISCOVERED occurs when a low energy Bluetooth GATT service is discovered, and ACTION_DATA_AVAILABLE occurs when data are received from the low energy Bluetooth device. The intent that allows for real-time data updates is the ACTION_DATA_AVAILABLE intent. If data are available, it is stored into a byte array named "value" using the getValue()

method and the byte array is passed to the getData() method where the data are processed.

In the getData() method, an if statement checks if the length of the packet is equal to 6, the number of expected bytes. This is shown in Figure 5-7. The bytes are then converted into unsigned 8-bit integers by performing a bitwise and function with 255. The order of the packet remains the same as the order as was shown in Chapter 3 where h stands for the heel load value, m stands for the medial load value, and l stands for the lateral load value. The “1” after the letter represents that the variable contains the lower 8-bits of the 14-bit load value and the “2” represents that the variable contains the upper 6-bits.

The two bytes are then recombined into a 14-bit variable using the combinebytes() method, shown in Figure 5-8. This method performs a bitwise shift operation by taking the value of the second byte of each load value and converting it to the value it would have if it were shifted left by 8-bits, shown in Figure 5-9. For example, assume that the total number of heel counts equals 1,000. The 14-bit representation of 1,000 is 1111101000 and when broken into two bytes, h1 = 11101000 and h2 = 00000011. When converted to integers, h1 = 232 and h2 = 3. Now in order to recombine these two integers back into its original 14-bit representation, h2 needs to be converted into binary, left shifted 8-bits using a bit wise operation, converted back into an integer, and added to h1. In this case, when h2 is shifted left 8-bits it equals 1100000000 or 768. By adding the shifted h2 back to h1, 232 + 768 = 1,000, the original value of 1,000 is obtained. Since the original data structure is 14-bits, the maximum value h2 can reach is 00111111 or 63.

Therefore, there are 64 conversions necessary in order to cover the entire range of possible values (0 through 63) that h2 can have.

In order to convert the number of counts into pounds, v2 is converted into a double type variable by multiplying by 1.0 as shown in Figure 5-10. The number of counts is then multiplied by the ratio of pounds to counts (250 lbs. /16384 counts). The two bytes for the heel, medial, and lateral counts were all combined and converted into a load value using the combinebytes() method and were stored in the variables ht, mt, and lt, respectively. The total load was calculated by adding ht, mt, and lt together and was stored in the variable tf. After storing the total force and the three individual forces, the variables containing the byte values were reset back to zero for the next reading as shown in Figure 5-11.

Determination of the Max Force for Each Step

After the total force between all three sensors is calculated and stored in the variable tf, the maximum force of each step is determined by comparing all values measured during a step. A beginning of a step is defined when total force first exceeds 10% of the subject's body weight and the end of the step is reached when the total force drops below 10% as shown in Figure 5-12.

The total force is compared with mf_local and if tf is greater than mf_local, then mf_local is set equal to tf. If the total force is greater than 10% of the subject's body weight, then the step_condition is set equal to 1 and it enables the code to look for when the total force drops down below 10% of subject body weight to mark the end of the step, shown in Figure 5-13. The variable mf is used to store the maximum value of each step for data collection.

The end of the step is defined when the step condition is equal to 1 and the total force is less than 10%. At the end of the step, the value contained in the variable mf is the maximum force of the step and is graphed onto the five step history bar graph. The value of mf is compared to the upper and lower boundary, which in this case is 30% and 20% body weight, respectively. If mf is greater or equal to the upper boundary, the total number of red steps (overloaded steps) is increased. If mf is less than the upper boundary but greater than or equal to the lower boundary, then the number of green steps (acceptable steps) is increased. If mf is less than the lower boundary, then the number of gray steps (under loaded steps) is increased. At the end of the step, the counter of total number of steps is increased by one and the step condition is set back to zero.

Graphing

Once the data are received from the Atlas module and processed, it is ready to be graphically displayed to the user. The two ways that these data are displayed to the end user are through the bar graph and a five step history bar graph.

The bar graph is used to display the current total load exerted on the Atlas sensors by the subject, shown in Figure 5-14. This bar graph updates at a rate of 20 times per second and changes colors when weight thresholds are crossed. These weight thresholds are the upper and lower boundaries of 20% and 30% body weight, respectively. When the current total load is below 20% body weight, the color of the bar is gray. When the load is within 20% and 30%, the bar turns green to indicate to the subject that they are weight bearing within the acceptable

load range. The bar turns red when the load exceeds 30% to indicate to the subject that they are overbearing.

The history bar graph is used to display the maximum load of the 5 previous steps to the subject. It is updated when at the end of a step. The color of the bars on the history bar graph follow the same color scheme as the bar graph above.

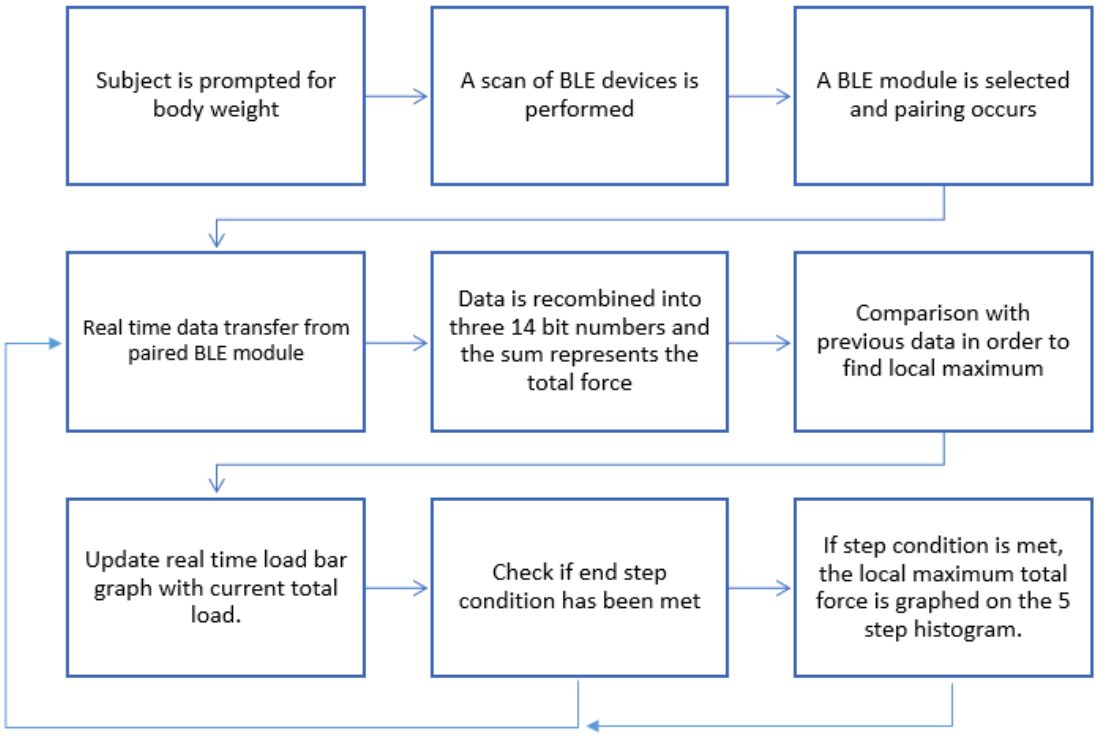

Figure 5-1. High level system flow chart for Atlas Android App code

Figure 5-2. Prompting the subject for their weight

Figure 5-3. A screenshot of the app showing a list of available BLE devices in the area

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_scan:
            mLeDeviceListAdapter.clear();
            scanLeDevice(true);
            break;
        case R.id.menu_stop:
            scanLeDevice(false);
            break;
    }
    return true;
}
```

Figure 5-4. Code executing a new scan when the Scan menu option is selected

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    final BluetoothDevice device = mLeDeviceListAdapter.getDevice(position);
    if (device == null) return;
    final Intent intent = new Intent(this, BarGraph.class);
    intent.putExtra(DeviceControlActivity.EXTRAS_DEVICE_NAME, device.getName());
    intent.putExtra(DeviceControlActivity.EXTRAS_DEVICE_ADDRESS, device.getAddress());
    if (mScanning) {
        mBluetoothAdapter.stopLeScan(mLeScanCallback);
        mScanning = false;
    }
    startActivity(intent);
}
```

Figure 5-5. The activity Bargraph.java is started when a BLE device is selected

```
private final BroadcastReceiver mGattUpdateReceiver = (context, intent) -> {
        final String action = intent.getAction();
        if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
            mConnected = true;
            updateConnectionState("Connected");
            invalidateOptionsMenu();
        } else if (BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
            mConnected = false;
            updateConnectionState("Disconnected");
            invalidateOptionsMenu();
            clearUI();
        } else if (BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
            // Show all the supported services and characteristics on the user interface.
            displayGattServices(mBluetoothLeService.getSupportedGattServices());
        } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
            String okay = intent.getExtras().getString(BluetoothLeService.okay);
            value = mNotifyCharacteristic.getValue();
            getData(value);

        }
};
```

Figure 5-6. The Broadcast Receiver handles updates from the connected BLE device

```
public void getData(byte[] value) {

    //convert the data packet into a string for display
    v1 = Arrays.toString(value);

    //check packet length
    packetlen = value.length;

    //if all data is present
    if(packetlen == 6) {

        //perform a bitwise and function with 255 to convert into unsigned 8 bit integer
        h1 = value[0] & 0XFF;
        h2 = value[1] & 0XFF;
        m1 = value[2] & 0XFF;
        m2 = value[3] & 0XFF;
        l1 = value[4] & 0XFF;
        l2 = value[5] & 0XFF;
```

Figure 5-7. The getData() method processes the data received from the Broadcast Receiver

```
//combination of both bytes into one double type number
ht = combinebytes(h1, h2);
mt = combinebytes(m1, m2);
lt = combinebytes(l1, l2);
```

Figure 5-8. The upper and lower bytes are sent to the combinebytes() method

```
public double combinebytes(int v0, int v3) {

    if (v3 == 0) {
        v2 = v0;
    } else if (v3 == 1) {
        v2 = v0 + 256;
    } else if (v3 == 2) {
        v2 = v0 + 512;
    } else if (v3 == 3) {
        v2 = v0 + 768;
    } else if (v3 == 4) {
        v2 = v0 + 1024;
```

Figure 5-9. Combining the upper and lower bytes into one double type number

```
    c = (1.0*(v2));

    d = c*250/16384;

    return d;
}
```

Figure 5-10. Converting the number of counts into pounds

```
//summation of all three sensor values
tf = ht + mt + lt;

//reset back to zero for next reading
h1 = 0;
h2 = 0;
m1 = 0;
m2 = 0;
l1 = 0;
l2 = 0;
```

Figure 5-11. Resetting all variables back to 0 for the next reading

```
if(tf > mf_local){
    mf_local = tf;

    if(tf>10){
        step_condition = 1;
    }

    if(mf_local > mf){
    mf = mf_local;
    }
}
```

Figure 5-12. Determination of the beginning of a step and finding local maximum

```
if(step_condition == 1) {          //////Step condition is met when a step is greater than 10%
    if(tf<10) {
        graph2LastXValue1 += 1d;
        stepSeries.appendData(new GraphView.GraphViewData(graph2LastXValue1, mf), true, 1600);

        if (mf >= top_g_boundary) {
            num_rsteps++;
        } else if (mf >= bottom_g_boundary) {
            num_gsteps++;
        } else {
            num_bsteps++;
        }

        num_steps++;
```

Figure 5-13. The end step condition is met when the when total force goes below 10% body weight

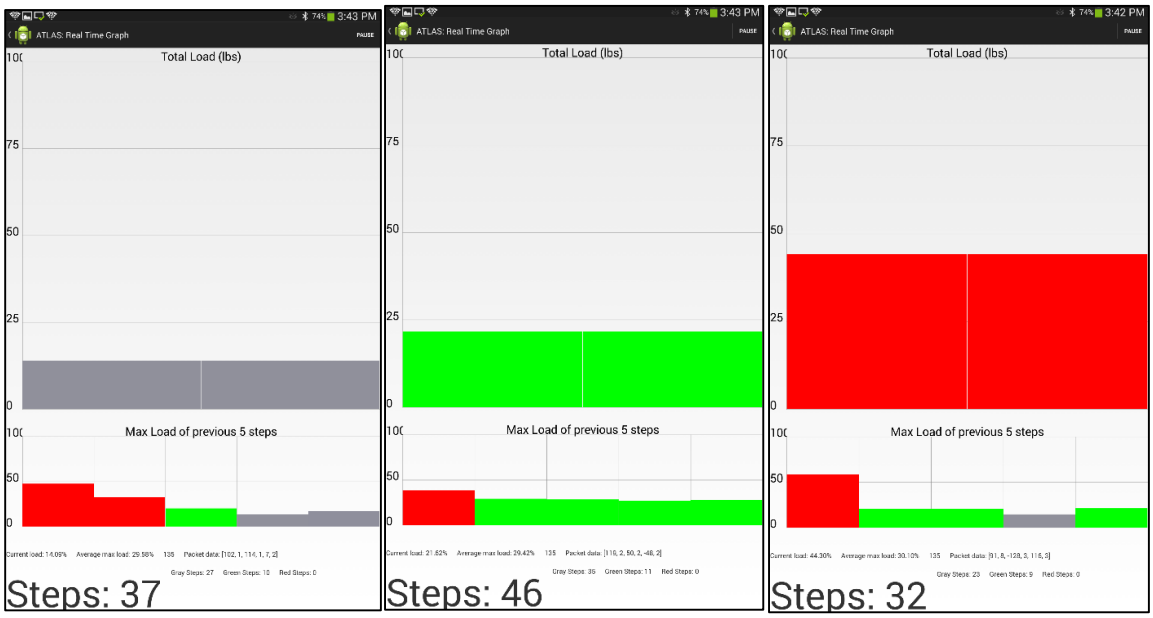

Figure 5-14. The total load bar graph and history bar graph of the Atlas App provides subjects real-time feedback. Three separate screens are shown. The top portion of the screen features a bar graph which changes based on the current load. The history bar graph below it shows the max load of the 5 previous steps.

CHAPTER 6

RESULTS

The summary of the data from the 10 subject study is shown in Tables 6.1 and 6.2. Each of the subjects performed the control and experimental procedures. The data are separated into two tables. Table 6-1 displays data from subjects who performed the control procedure first followed by the experimental procedure (Group 1) and Table 6-2 shows subjects which were subjected to the experimental procedure first (Group 2).

The average load of all the steps with and without feedback are shown in Figure 6-1. The average load without feedback was 30.03 ± 11.5 lbs. and the average load with feedback was 24.87 ± 2.9 lbs.

A one-way ANOVA was performed on the number of acceptable steps of the four test conditions: Group 1 no feedback, Group 1 feedback, Group 2 no feedback, and Group 2 feedback. The null hypothesis states there is no difference in means of the four populations. The resulting F-Ratio was calculated to be 4.54 and using an F-Distribution table we found $F_{crit}(3,16) = 2.46$. It is seen that the F-ratio is greater than the F-crit 4.54>2.46, which results in the rejection of the null hypothesis. From this, the conclusion that one of the population means is different from at least one other population mean can be drawn.

A histogram of steps without feedback is shown in Figure 6-2. The steps are sorted by the maximum load in terms of the percentage of the subject's body weight exerted during the step. There were 165 steps in the 20-29% body weight range with a wide distribution of steps. A histogram of steps with feedback is shown in Figure 6-3. There are 342 steps in the 20-29% body weight range with a much narrow distribution in comparison. Figure 6-4 compares the number of steps within 20-30% body weight for steps with and without feedback. Figure 6-5 shows the percentages of steps within 20-30% body weight for both Groups 1 and 2. In both groups, the percentages of steps were higher when feedback was provided.

Table 6.1. Data from subjects performing control procedure followed by experimental procedure

Control -> Experimental		Without feedback		With feedback	
Subject	Weight	Average load (lbs..)	Acceptable steps	Average load (lbs.)	Acceptable steps
1	140	54.3 ± 8.1	0	25.5 ± 5.4	35
4	139	28.7 ± 5.7	32	23 ± 5.7	40
7	153	26.4 ± 10.4	17	26.1 ± 9.7	28
8	172	36.5 ± 7.8	3	25.7 ± 4.8	39
10	170	17.1 ± 4.8	8	23 ± 5.1	40

Table 6-2. Data from subjects performing experimental procedure followed by the control procedure

Experimental -> Control		Without feedback		With feedback	
Subject	Weight	Average load (lbs..)	Acceptable steps	Average load (lbs.)	Acceptable steps
2	201	45.1 ± 5.2	0	31.9 ± 6.3	18
3	167	30.1 ± 5.6	18	26.4 ± 4.5	37
5	141	17.8 ± 5.4	12	23.8 ± 6.0	32
6	189	23 ± 6.7	38	21.3 ± 6.3	33
9	125	21.3 ± 3.3	37	22 ± 4.1	40

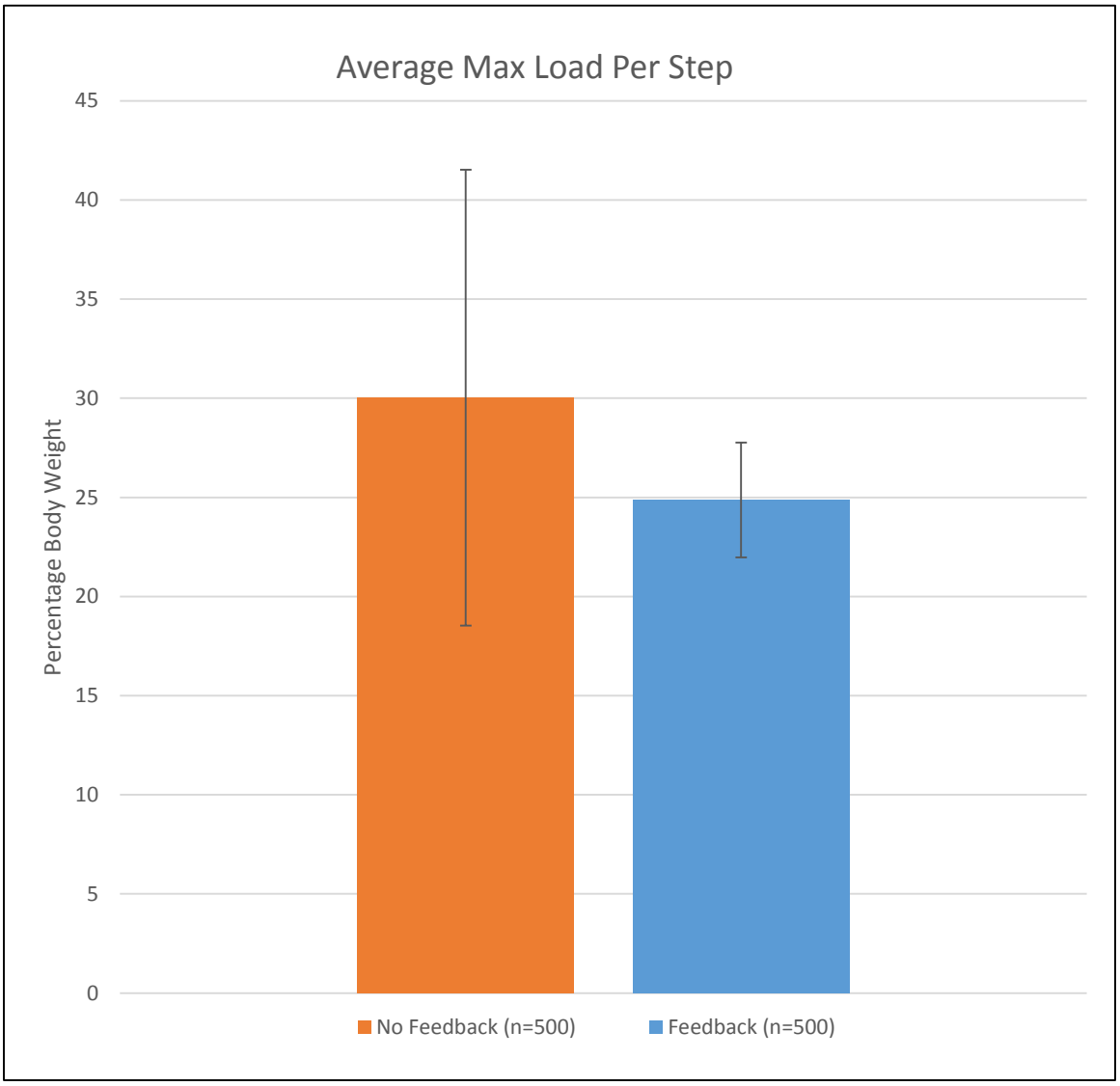

Figure 6-1. The average max load of per step is shown in the bar graph above for steps with and without feedback. The steps with no feedback (n=500) had an average of 30.03 ± 11.5 lbs. and the steps with feedback (n=500) had an average of 24.87 ± 2.9 lbs. A paired t-test gave a p-value of 0.1513.

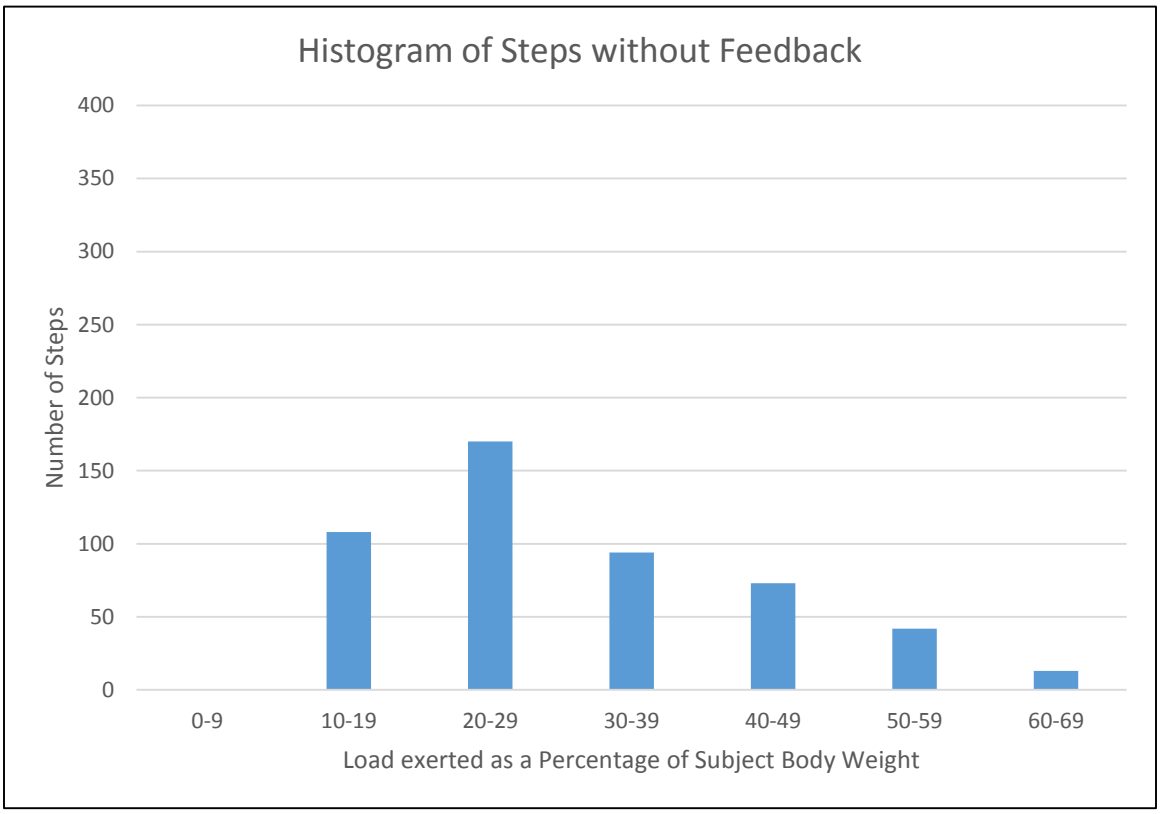

Figure 6-2. A histogram of the steps without feedback (n=500). The steps are sorted by the amount of max load exerted as a percentage of the subject's body weight. In the 20-29 range, there is a peak of 165 steps with a wide distribution surrounding the peak.

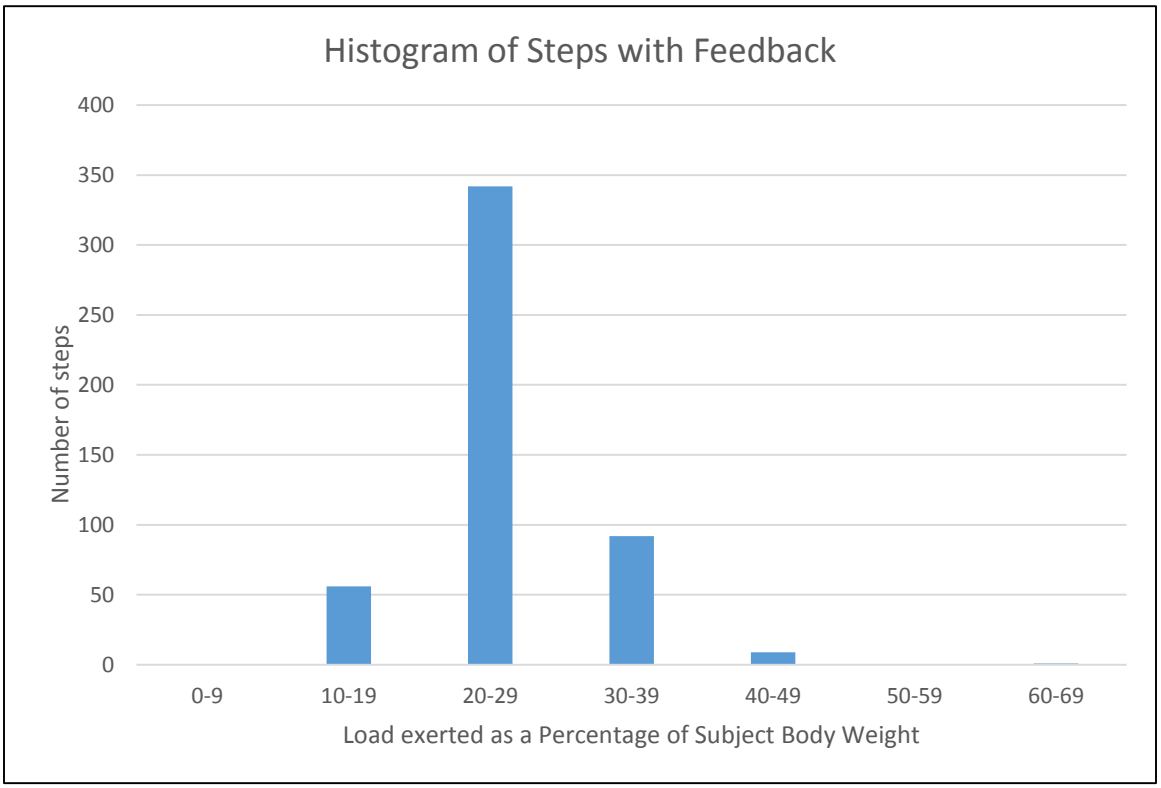

Figure 6-3. A histogram of the steps with feedback (n=500). The steps are sorted by the amount of max load exerted as a percentage of the subject's body weight. In the 20-29 range, there is a peak of 342 steps and the distribution is much narrower in comparison to the histogram of no feedback steps.

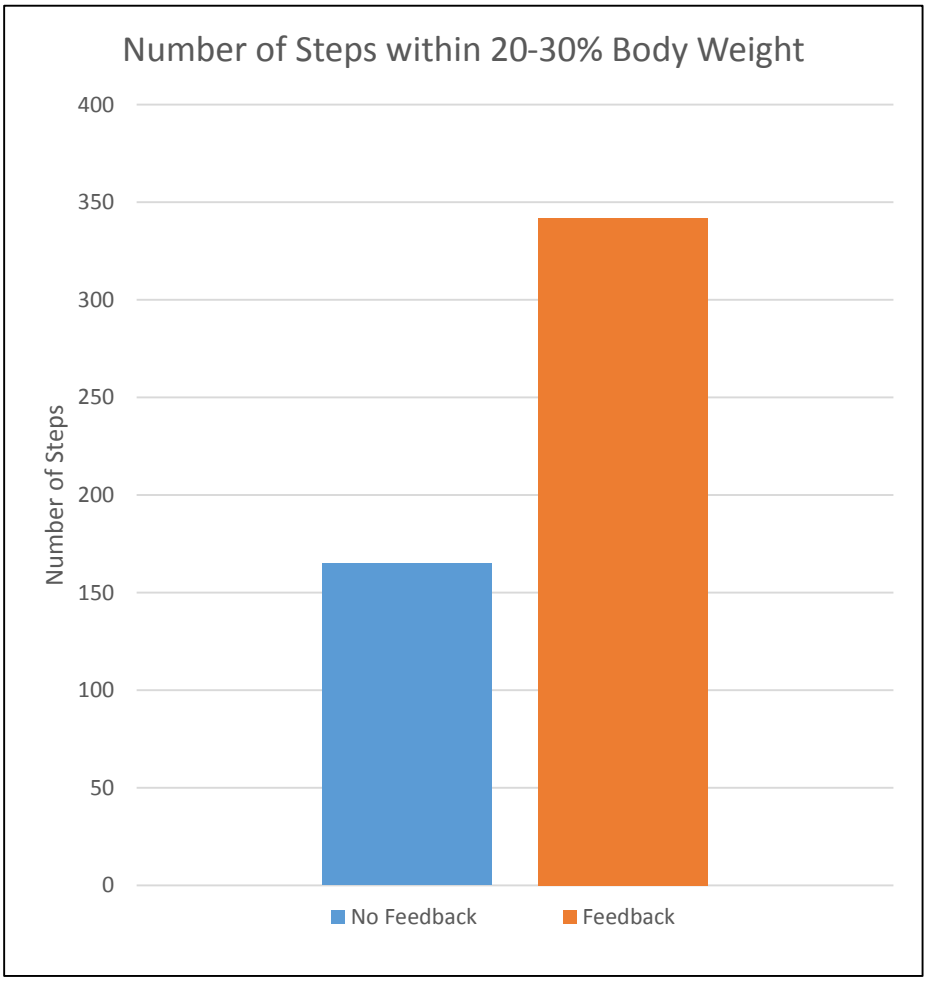

Figure 6-4. The number of steps within 20-30% body weight for steps with and without feedback. When feedback was not provided, 165 steps out of 500 were within 20-30% body weight. When feedback was provided, 342 out of 500 steps were within 20-30% body weight.

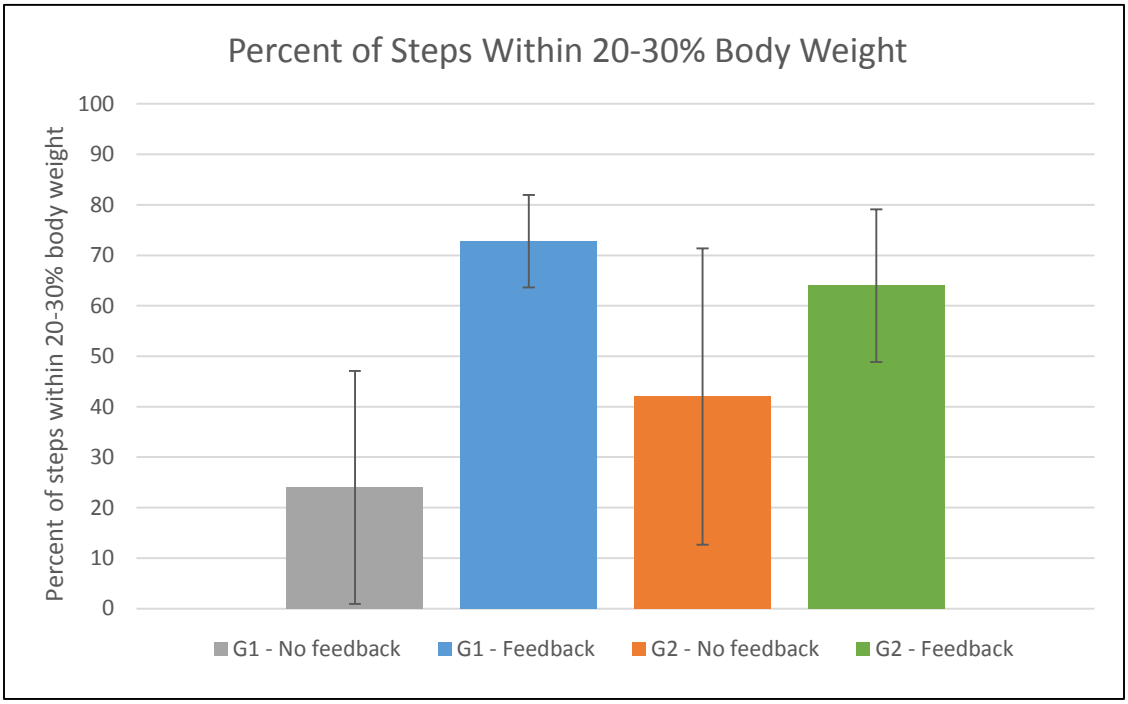

Figure 6-5. The percentages of steps within 20-30% body weight for both Groups 1 and 2. In both groups, the percentage of steps within 20-30% body weight was higher when feedback was provided.

CHAPTER 7

DISCUSSION

Results

The purpose of this experiment was to determine if real-time feedback had a statistically significant effect on a subject's ability to partially weight bear 20-30% of their body weight. The average load of the 50 steps with no feedback was 30.03 ± 11.5 lbs. This is a high average load and a large distribution compared to 24.87 ± 2.9 lbs. which was the overall average load of the 50 steps with feedback, shown in Figure 6-1. There is a clear improvement in the ability of a subject to partially weight bear at 20%-30% of their body weight when real-time feedback is provided.

Two paired t-tests were conducted in order to determine if the order of performing the control and experimental procedures had any effect on the subject's results. A first paired t-test was conducted on the 5 subjects (Group 1) assigned to perform the control procedure first. The values used for the paired t-test were the number of acceptable steps from the control procedure (no feedback) and the number of acceptable steps from the experimental procedure (with feedback). The difference of the means equaled -24.40 with a 95% confidence interval from -41.11 to 7.36. The two tailed P-value for this paired t-test equaled 0.0165, which by conventional criteria shows that this difference between the two sets of data is considered to be statistically significant. This indicates that the introduction of real-

time feedback had statistically significant impact on the patient's ability to partially weight bear at 25% of their body weight.

The second paired t-test was conducted on the 5 subjects that were assigned to perform the experimental procedure first (Group 2). The values that were used for the paired t-test were the number of acceptable steps from the control procedure (no feedback) and the number of acceptable steps from the experimental procedure (with feedback). The difference of the means equaled -11.00 with a 95% confidence interval from -25.08 to 3.08. The two tailed P value for this paired t-test equaled 0.0958, which by conventional criteria shows that the difference between the two sets of data is not considered to be statistically significant. This indicates that learning is taking place during the first 50 steps with feedback when the experimental procedure is performed first followed by the control procedure and this learning effect has a significant impact on the next 50 steps without feedback.

Observations

During the experimental testing procedure, it was observed that subjects were much more confident in their ability to partially weight bear within 20 to 30% of their body weight when receiving real-time feedback from the Atlas Android App. Subjects noted they felt the feedback from the app made it much easier to weight bear within the target load range compared to when no feedback was provided. In one particular instance, one subject noted that they felt that they could consistently weight bear 20-30% of their body weight after performing the 50 steps with feedback from the Atlas Android app.

Future Work

This experiment was designed as a pilot study with the intent of gathering population data in order to statistically power a larger study. The purpose of the larger study would be to study the effects of real-time feedback on a subject's ability to partially weight bear in a larger experimental group. Other future additions may include expanding the scope of the experiment to examine at the effect of real-time feedback at other commonly prescribed partially weight bearing ranges.

This experiment examined the effects of real-time feedback on partial weight bearing. However, different mediums of providing real-time feedback should be explored as a part of future studies. Such studies would look at the variance of compliance rates when different types of feedback are provided as well as the efficacy real-time feedback has on affecting the subject's ability to partially weight bear.

APPENDIX A

EXPERIMENTAL PROTOCOL

Investigation of Effectiveness of Real-time Feedback Assisted Partial Weight Bearing

BACKGROUND AND INTRODUCTION

Study Objective

The objective of this study is to determine whether visual real-time load feedback can improve a person's ability to control partial weight bearing while ambulating with crutches.

Background

When a limb is immobilized in a cast for long periods of time after surgery, the rate of bone loss begins to exceed the rate of bone formation, especially if there is no load exerted on the limb.[1] This leads to bone density loss and decreases the strength of the bone.[1] Research in orthopedic practice has shown that controlled cyclic loading can help accelerate the healing of bone and counteract bone density loss.[1],[2]

This controlled cyclic loading can be accomplished by limiting the loading of the limb to a fraction of the full body weight – a practice called Partial Weight Bearing (PWB). Patients are instructed to limit the load on the healing limb during walking to PWB values prescribed by the physician. Commonly prescribed PWB values include 25%, 50%, and 75% of the patient's body weight. The patients are prescribed lower PWB values initially and these values are increased as the rehabilitation progresses. A problem with current PWB practice is that patients' ability to determine actual load and limit it to a target value is limited and tends to exceed the prescribed PWB values, which can result in nonunion of the fractured bone.[3]

The current standard of care in prescribing PWB is to have patients learn how to control the load in the injured limb by pressing on a weighing scale and memorizing the "feel". Patients are expected to memorize perception of that load and recall the loading during

ambulation with crutches, but without the benefit of looking at the scale. This practice is insufficient and leads to the patients over or under loading.[3]

In order to measure actual limb load data in patients on PWB prescription, an Ambulatory Tibial Load Analysis System (ATLAS) was developed. The ATLAS system provides medical practitioners with the ability to collect and analyze limb loading data in patients with lower extremity fractures, specifically with tibial fractures, during the rehabilitation period. ATLAS uses a Don Joy Orthopedics MaxTrax CAM Walker, which is equipped with three pressure sensors, as shown in Figure 1. The pressure sensors are embedded in a foam pad and are positioned under the heel, medial, and lateral areas of the foot, Figure 2. Attached on the side of the boot is a wireless data recorder that stores, and communicates in real-time, the load data to a mobile data display, in this case, a tablet computer. ATLAS is a minimal risk device and has previously been cleared by the IRB for use in other studies: (IRB_00057602) and (IRB_00058956).

The display module uses an Android OS-based application for transferring load data from the ATLAS-equipped CAMWalker. Figure 3. The display serves as a load meter that shows the user how much total load they are placing on the limb. The load meter changes between gray, green, or red depending whether the total load is below, within, or above the specified target range. The bar graph shows the user the maximum total force for their previous steps and uses the same color code as the load meter. This display will be used to provide subjects with real-time feedback. While patients are receiving feedback from this display, they will be focusing on keeping the maximum total load of each step within the region in which the total load bar turns green. For this study the green region will be 20-30% of the subject's body weight.

PARTICIPANT SELECTION CRITERIA

Inclusion Criteria:

- 18-70 years old
- No lower extremity injuries affecting normal walking

Exclusion Criteria:

- Less than 18 years old.
- Older than 70 years old.
- Individuals with any condition that affects normal walking

DESIGN

Observational.

This study is a minimal risk pilot study of real-time feedback-assisted PWB ambulation in healthy human volunteers. This is not a clinical trial.

STUDY PROCEDURES

Recruitment/ Participant Identification Process:

The research team will recruit University of Utah students, faculty, and staff. Recruitment will be done by informing participants of this study in person and placing informational flyers around the University of Utah Campus.

Consent process:

All prospective participants will receive detailed information about the study from research personnel trained in the protocol before giving consent. If interested in participating, subjects will read and understand the consent form, which they will sign if they choose to participate. The subjects will be able to ask the researchers any questions they have about the study. A copy of the consent form will be provided to the subject and the original will be archived by the investigator.

Procedures:

After consent is received from the subject, he/she will be asked to stand on a scale in order to obtain the subject's body weight. This body weight will be used to calculate the acceptable load range which will be determined by multiplying the body weight of the subject by (20%) to obtain the lower end of the target range and by (30%) in order to obtain the higher end.

The subject will be then be asked to place and secure their foot into ATLAS. PWB requires the subject to walk with crutches, so they will be trained by a research team member on how to walk with crutches using 3-point gait. The participant is trained on how PWB with crutches, a pair of red-dot adult Guardian crutches will be adjusted to fit the subject. The subject will then be allowed to practice walking using 3-point gait until they signal that they are comfortable with the procedure. The participants will be randomized into two groups. One group will perform the control procedure followed by the experimental procedure, and the other group will perform the experimental procedure followed by the control procedure.

Control Procedure:

While wearing ATLAS, the subject will be asked to partially bear 25% of their total body weight on the ATLAS-wearing-foot. Once the subject can consistently PWB 25% of their body weight for several seconds, the subject will be asked to memorize how to PWB at 25% body weight.

The subject will then be asked to walk 50 steps on even terrain with crutches using 3-point gait while trying to replicate PWB 25% during each step. At no time during the control procedure will the subject receive any feedback from the Atlas app.

Experimental Procedure:

While wearing ATLAS, the subject will be asked to exert 25% of their body weight on the ATLAS wearing foot onto a weight scale using crutches to PWB. Once the subject can consistently PWB 25% of their body weight for several seconds, the subject will be asked to memorize how to PWB 25%.

The subject will then be asked to walk 50 steps on even terrain with crutches using 3-point gait while trying to replicate PWB 25% and also receiving real-time feedback from the Atlas app. A researcher will walk in front of the subject holding the android device. The display will be masked from the researcher's view in order to prevent the researcher from conveying any additional cues to the subject.

STATISTICAL METHODS, DATA ANALYSIS AND INTERPRETATION

Definitions

Acceptable step: A step is defined as acceptable if it is between 20-30% of the subject's body weight

Null and Alternative Hypothesis:

H_0: There is no significant difference in the number of acceptable steps when feedback is provided and when no feedback is provided

H_i: There is a significant difference in the number of acceptable steps when feedback is provided and when no feedback is provided

Sample size:

Number of Participants: We are gathering data in order to gather data in order to determine population characteristics for future studies. We will plan to enroll up to 12 subjects. 5 subjects will perform the control procedure first followed by the experimental procedure, 5 subjects will perform the experimental procedure followed by the control procedure, and the 2 remaining subjects will only be enrolled in the case that any participants wish to discontinue their participation.

Number of steps: Participants will be asked to advance 50 steps with feedback and 50 steps without feedback. This will result in 50 data points from each procedure. These data points will be used to calculate the rate of compliance for individual subjects.

Data analysis and Interpretation:

The control and experimental data of each subject will be compared using a paired t-test in order to prove or disprove the null hypothesis that there is significant difference between the two data sets.

APPENDIX B

CONSENT DOCUMENT

BACKGROUND

The purpose of this study is to determine if a subject's ability to partially weight bear within a target load range during crutch-assisted walking significantly increases when receiving real-time feedback from a mobile app displaying the total load exerted on the subject's foot as compared to crutch-assisted walking with no feedback from the mobile app.

When a limb is immobilized in a cast for long periods of time, the rate of bone loss begins to exceed the rate of bone formation when there is no load exerted on the limb. This leads to bone density loss and decreases the strength of the bone. Research in orthopedic practice has shown that controlled cyclic loading can help accelerate the healing of bone and also counteract bone density loss. When partial weight controlled cyclic loading is done, it is called Partial Weight Bearing or PWB. Patients are instructed to PWB on the healing limb with PWB values prescribed by the physician. Commonly prescribed PWB values include 25%, 50%, and 75% of the patient's body weight. The patients are prescribed lower PWB values initially and these values are increased as the rehabilitation period progresses.

The benefits of PWB on fracture rehabilitation is supported by multiple studies. Studies show that controlled loading of a healing limb promotes bone growth, which helps accelerate the healing of a fracture. A problem with current PWB practice is that it is difficult for patients to maintain the prescribed PWB loading values during walking, which can result in nonunion of the fractured bone.

By doing this study, we are investigating whether real-time feedback allows for the subject to partially weight bear within a target load range more reliably compared to maintain the loading without any feedback.

STUDY PROCEDURE

If you agree to participate in this study, you will be asked to walk 50 steps while using crutches and wearing a standard boot cast that has load sensors that measure how much load you are placing on your leg. The information collected by this sensor is transmitted wirelessly to an Android device using Bluetooth. This Android device will display your current load in real-time as well as the maximum load of your five previous steps.

This study will take you no more than 1.5 hours to complete. First you will be asked to stand on a scale in order to obtain your body weight. Next, a researcher will assist you in putting on the ATLAS boot onto your right foot. You may walk around to ensure that the boot straps are adjusted to fit your foot appropriately. As part of this study, you will be asked to partially weight bear on your right foot by using crutches. The researcher will train you on how to partially weight bear with crutches, you will be allowed to practice walking around with crutches until you feel comfortable.

During the study, you will be asked to first partially weight bear without feedback and then with feedback. You will be randomly assigned to start either with feedback, or without feedback fist. The procedure for no feedback and feedback are written below. The researcher will tell you which procedure you will be doing first.

<u>Without Feedback:</u>

You will be asked to put 25% of your body weight using crutches on your foot (with the ATLAS boot on) onto a weight scale. Once you can consistently maintain a reading of 25% of your body weight for several seconds, you will be asked to remember how much force you were putting on your foot.

You then will be asked to walk 50 steps down a hallway while trying to put 25% of your body weight down on your foot every time you step down on the foot wearing ATLAS.

With Feedback:

You will be asked to put 25% of your body weight using crutches on your foot (with the ATLAS boot on) onto a weight scale. Once you can consistently maintain a reading of 25% of your body weight for a couple of seconds, you will be asked to remember how much force you were putting on your foot.

The real-time feedback will be shown to you on an android tablet. It consists of a load meter and a bar graph which shows the total force of your past 5 steps.

You then will be asked to walk 50 steps down a hallway while trying to put 25% of your body weight every time you step down on your right foot. A researcher will walk in front of you while holding the tablet in order to give you real-time feedback.

RISKS

The risks of this study are minimal. You may feel uncomfortable thinking about or talking about personal information related to your body weight. These risks are similar to those you experience when discussing personal information with others. If you feel upset from this experience, you can tell the researcher, and he/she will tell you about resources available to help. You may feel tired after completing the required tasks. You may experience minor bruising or abrasions resulting from wearing the boot cast. There may also be risks that are currently unforeseeable.

BENEFITS

There are no direct benefits for taking part in this study. However, we hope the information we get from this study may help develop a greater understanding of improving partial weight bearing in the future.

ALTERNATIVE PROCEDURES

There are no alternative procedures. You may decline to participate in this study. If you do not wish to participate in the study, you may opt out at any time by notifying the researcher.

CONFIDENTIALITY

We will keep all research records that identify you private to the extent allowed by law. A social security number is not required to participate in this study. Records about you will be kept on a password protected computer. Only those who work directly on this study will be allowed access to your information.

However, if you disclose information that gives study staff a reason to believe that a child or disabled or elderly adult has been subjected to abuse or neglect, study staff will report that information to Child Protective Services, Adult Protective Services, or the nearest law enforcement agency to the extent required by law.

There are some cases in which a researcher is obligated to report issues, such as serious threats to public health or safety.

The study may also be inspected as permitted by law, or by the FDA.

The results of the study will be published, but in such case only no personally identifiable information will be reported.

PERSON TO CONTACT

If you have questions, complaints or concerns about this study, you can contact Alvin Le at 208-371-7344, 24 hours. If you feel you have been harmed as a result of participation, please call Tomasz Petelenz PhD at 801-201-4746 who may be reached on Monday – Friday from 8am – 5pm,

Institutional Review Board: You may contact the Institutional Review Board (IRB) if you have questions regarding your rights as a research subject. Also, you may contact the IRB if you have questions, complaints or concerns which you do not feel you can discuss with the investigator. The University of Utah IRB may be reached by phone at (801) 581-3655 or by e-mail at irb@hsc.utah.edu.

Research Participant Advocate: You may also contact the Research Participant Advocate (RPA) by phone at (801) 581-3803 or by email at participant.advocate@hsc.utah.edu.

VOLUNTARY PARTICIPATION

Your participation in this study is entirely voluntary. You can tell us that you don't want to be in this study. You can start the study and then choose to stop your participation in the study later. This will not affect your relationship with the investigator or have any effect on your student standings.

COSTS AND COMPENSATION TO PARTICIPANTS

There will be no costs to you related to your participation in the study.

CONSENT

By signing this consent form, I confirm I have read the information in this consent form and have had the opportunity to ask questions. I will be given a signed copy of this consent form. I voluntarily agree to take part in this study.

_____

Printed Name of Participant

_____ _____

Signature of Participant Date

_____

Printed Name of Person Obtaining Consent

_____ _____

Signature of Person Obtaining Consent Date

APPENDIX C

MODIFICATIONS TO THE ATLAS MICROCONTROLLER CODE

This section contains code that was added to the ATLAS MSP430F5529 firmware to allow for data to be transmitted to the BLE113 module. After data is sampled from the load sensors, the three 14-bit variables are broken down into two 8-bit variables and passed into UCA0TXBUF for transmission to the BLE113 module via UART.

```
void
init_uart(){
        P3SEL |=- it3;                  // P3.3 = USCI_A1 TXD

       //Direction of-bit 3 to output (1),
        P3DIR |= bit4 | bit3;


        UCA0CTL1 |= UCSWRST;             // **Put state machine in reset**
        UCA0CTL1 |= UCSSEL_2;            // SMCLK
        UCA0BR0 = 7;                     // 1MHz 115200 (see User's Guide)
        UCA0BR1 = 0;                     // 1MHz 115200 Setting Baud Rate
        UCA0MCTL |= UCBRS_1 + UCBRF_0;   // Modulation UCBRSx=1, UCBRFx=0
        UCA0CTL1 &= ~UCSWRST;            // **Initialize USCI state machine**


       // P3DIR |= bit2;                // Set the LED to output P1.
```

```
            UCA0IE |= UCRXIE;
}
void
send_uart(volatile sample_t *sample)
{
    P3OUT ^= bit4;  // Toggle Flow Control
    P3OUT ^= bit4;

    uint16_t heel16 = sample->heelCounts;
    uint8_t h1 = heel16;
    uint8_t h2 = (heel16>>8);

    //uint16_t medial16 = 4500;
    uint16_t medial16 = sample->medialCounts;
    uint8_t m1 = medial16;
    uint8_t m2 = (medial16>>8);
    //uint16_t lateral16 = 0;
    uint16_t lateral16 = sample->lateralCounts;
    uint8_t l1 = lateral16;
    uint8_t l2 = (lateral16 >> 8);

        while (!(UCA0IFG&UCTXIFG));      //Wait UART to finish before next send
        UCA0TXBUF = h1;
        while (!(UCA0IFG&UCTXIFG));      //Wait UART to finish before next send
        UCA0TXBUF = h2;
        while (!(UCA0IFG&UCTXIFG));      //Wait UART to finish before next send
        UCA0TXBUF = m1;
        while (!(UCA0IFG&UCTXIFG));      //Wait UART to finish before next send
```

```
        UCA0TXBUF = m2;
        while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
        UCA0TXBUF = l1;
        while (!(UCA0IFG&UCTXIFG));       //Wait UART to finish before next send
        UCA0TXBUF = l2;
}
```

APPENDIX D

BLE113 BLUETOOTH MODULE CODE

This section contains BLE113 code that allows for data to be received from the MSP4305529 and broadcasted to a paired device.

Hardware.xml

The Hardware.xml sets parameters such as UART baud rates and also pin interrupt initialization.

```
<?xml version="1.0" encoding="UTF-8" ?>

<hardware>
    <sleeposc enable="true" ppm="30" />
    <usb enable="false" />
    <sleep enable="false" />
    <txpower power="15" bias="5" />
    <script enable="true" />
    <usart channel="1" alternate="1" baud="115200" endpoint="none" flow="false"/>

    <!-- P0_0, P0_1, and P1_6 will be used for interrupts. Port 2 interrupts are not supported. -->
    <port index="0" pull="down" />

</hardware>
```

Gatt.xml

Gatt.xml defines the GATT profile of BLE113 module. The data being transmitted is stored in xgatt_data which has a characteristic UUID of e7add780-b042-4876-aae1-112855353dd1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

  <service uuid="1800">
    <description>Generic Access Profile</description>

    <characteristic uuid="2a00">
      <properties read="true" const="true" />
      <value>Atlas BLE</value>
    </characteristic>

    <characteristic uuid="2a01">
      <properties read="true" const="true" />
      <value type="hex">4142</value>
    </characteristic>
  </service>
  <service uuid="1234" advertise="true">
      <description>Cable replacement service</description>

      <characteristic uuid="e7add780-b042-4876-aae1-112855353dd1"
id="xgatt_data">
          <description>Data</description>
          <properties write="true" notify="true"/>
          <value variable_length="true" length="20" type="user" />
      </characteristic>
  </service>
</configuration>
```

Script.bgs

Script.bgs controls the lifecycle of the BLE113 module during operation. The system_boot event enables interrupts on port P0.0, which enables one-way flow control. The hardware_io_port_status event toggles the RX watermark to enable the reception of incoming data. The system_endpoint_watermark_rx event receives the transmitted data and broadcasts the data using attributes_write, which writes the data to the characteristic xgatt_data.

```
dim result
dim in(20) # endpoint data in
dim in_len
dim out(20) # endpoint data out
dim out_len


event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)
    call system_endpoint_set_watermarks(5, 0, 0) # disable watermarks
    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)

    # enable interrupt on P0_0 and P0_1 rising edge
    # (parameters are port=0, bitmask=0b00000011, edge=rising)
    call hardware_io_port_config_irq(0, 3, 0)

end

# catch button press for P0_0 (active HIGH configuration, hardware.xml pulls it low)
event hardware_io_port_status(delta, port, irq, state)
    if port = 0 then
        if (irq & 1) = 1 then
            # P0_0 is HIGH and the source of this interrupt
            # DEVKIT UART OUTPUT: P0_0 pressed
            call system_endpoint_set_watermarks(5, 1, $ff) # set RX watermark
```

```
            call system_endpoint_set_watermarks(5, 0, $ff) # set RX watermark
        end if
    end if
end

event system_endpoint_watermark_rx(endpoint, size)

    if endpoint = 5 then
        in_len = size
        if in_len > 20 then
            in_len = 20
        end if

        call system_endpoint_rx(5, in_len)(result, in_len, in(0:in_len))
        call attributes_write(xgatt_data, 0, in_len, in(0:in_len))

    end if
end

event connection_disconnected(connection, reason)
    call system_endpoint_set_watermarks(5, 0, 0) # disable watermarks

    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)
end
```

APPENDIX E

ATLAS ANDROID APP CODE

AndroidManifest.xml

The Android Manifest contains essential information about the app to the Android system. The manifest includes activities as well as permissions allowing for Bluetooth capabilities.

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.example.app" >


    <uses-feature android:name="android.hardware.bluetooth_le"
android:required="true"/>




    <uses-permission android:name="android.permission.BLUETOOTH"/>

    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"/>


    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />


    <application

        android:allowBackup="true"

        android:icon="@drawable/ic_launcher"

        android:label="@string/app_name”
```

```
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.app.InformationActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait">
        </activity>

        <activity android:name=".DeviceScanActivity"/>
        <activity android:name=".DeviceControlActivity"/>
        <activity
            android:name=".RealTimeGraphing"
            android:screenOrientation="portrait">
        </activity>
        <activity
            android:name=".BarGraph"
            android:screenOrientation="portrait">
        </activity>
        <service android:name=".BluetoothLeService" android:enabled="true"/>
    </application>

</manifest>
```

InformationActivity.java

The InformationActivity.java is the activity that is initialized when the app first starts. This screen allows for the user to enter in their body weight and then scan for available Bluetooth devices. When the “Scan for devices” button is clicked, the value in the edit view is passed into the MainActivity,java activity and the MainActivity.java is launched.

```
package com.example.app;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.TextView;


public class InformationActivity extends Activity {


    public String weight;


    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        // Sets up UI references.


        final TextView weightText =(TextView)findViewById(R.id.weightedit);


        Button devicebutton= (Button) findViewById(R.id.findevicesbtn);


        devicebutton.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
        public void onClick(View v) {


            weight = String.valueOf(weightText.getText());
            final Intent intent = new Intent(InformationActivity.this,
MainActivity.class);
            intent.putExtra("subject_weight", weight);
            startActivity(intent);
        }
    });
  }
}
```

activity_main.xml

This is the resource file for InformationActivity.java. It consists of an edit text view for entering the user's weight, a text view prompting the user for their weight in pounds, and a button which when clicked starts the MainActivity.java.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:ems="10"
        android:id="@+id/weightedit"
        android:layout_marginTop="50dp"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/weighttext" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Subject Weight (lbs.): "
        android:id="@+id/weighttext"
        android:layout_alignBottom="@+id/weightedit"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="5dp"/>

    <Button
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Find Devices"
        android:id="@+id/findevicesbtn"
        android:layout_marginTop="95dp"
        android:layout_below="@+id/weightedit"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

MainActivity.java

The MainActivity.java is launched when the user scans for available Bluetooth devices. This activity lists all available Bluetooth devices in the area as well as allows for the user to perform additional scans, if necessary. The devices listed in this activity can be clicked. Upon being clicked, the BarGraph.java activity is launched and the Android devices is connected with the selected Bluetooth device.

```
package com.example.app;


import android.app.Activity;
import android.app.ListActivity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.os.Handler;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;


import java.util.ArrayList;


public class MainActivity extends ListActivity {
```

```
//Initialization
private LeDeviceListAdapter mLeDeviceListAdapter;
private BluetoothAdapter mBluetoothAdapter;
private boolean mScanning;
private Handler mHandler;

//Set permission
private static final int REQUEST_ENABLE_BT = 1;
// Stops scanning after 10 seconds.
private static final long SCAN_PERIOD = 10000;
public String weight;

//Checks for bluetooth availability on creation of this activity
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getActionBar().setTitle(R.string.title_devices);
    mHandler = new Handler();

    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        weight = extras.getString("subject_weight");
    }


    // Use this check to determine whether BLE is supported on the device.
Then you can
    // selectively disable BLE-related features.

    //If BLE not supported, print "BLE is not supported", else do nothing here
```

```
        If
(!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUE
TOOTH_LE)) {
            Toast.makeText(this, R.string.ble_not_supported,
Toast.LENGTH_SHORT).show();
            finish();
        }


        // Initializes a Bluetooth adapter.  For API level 18 and above, get a
reference to
        // BluetoothAdapter through BluetoothManager.
        final BluetoothManager bluetoothManager =
                (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);
        mBluetoothAdapter = bluetoothManager.getAdapter();


        // Checks if Bluetooth is supported on the device.
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, R.string.error_bluetooth_not_supported,
Toast.LENGTH_SHORT).show();
            finish();
            return;
        }
    }


    //Creates option menu with scan and stop button. There is also a menu refresh
button
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        if (!mScanning) {
            menu.findItem(R.id.menu_stop).setVisible(false);
```

```
            menu.findItem(R.id.menu_scan).setVisible(true);
            menu.findItem(R.id.menu_refresh).setActionView(null);
        } else {
            menu.findItem(R.id.menu_stop).setVisible(true);
            menu.findItem(R.id.menu_scan).setVisible(false);
            menu.findItem(R.id.menu_refresh).setActionView(
                    R.layout.actionbar_indeterminate_progress);
        }
        return true;
    }


    //When scan is selected in the menu, clear the items in the adapter and then run the scanLeDevice() method by passing the boolean true into it
    //This method scans for BLE enabled devices in the vicinity and displays them in a list
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_scan:
                mLeDeviceListAdapter.clear();
                scanLeDevice(true);
                break;
            case R.id.menu_stop:
                scanLeDevice(false);
                break;
        }
        return true;
    }


    //After the activity is stopped and resumed again, the activity checks for bluetooth, and enables the list view adapter
```

```
    @Override
    protected void onResume() {
        super.onResume();

        // Ensures Bluetooth is enabled on the device.  If Bluetooth is not currently
enabled,
        // fire an intent to display a dialog asking the user to grant permission to
enable it.
        if (!mBluetoothAdapter.isEnabled()) {
            if (!mBluetoothAdapter.isEnabled()) {
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
            }
        }

        // Initializes list view adapter.
        mLeDeviceListAdapter = new LeDeviceListAdapter();
        setListAdapter(mLeDeviceListAdapter);
        scanLeDevice(true);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // User chose not to enable Bluetooth.
        if (requestCode == REQUEST_ENABLE_BT && resultCode ==
Activity.RESULT_CANCELED) {
            finish();
            return;
        }
        super.onActivityResult(requestCode, resultCode, data);
```

```
    }

    //When the activity is paused, stop scanning and clear the list of devices
    @Override
    protected void onPause() {
        super.onPause();
        scanLeDevice(false);
        mLeDeviceListAdapter.clear();
    }

    //When an item is clicked, get the device name and address and send it to
DeviceControlActivity.Class and start that new activity
    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        final BluetoothDevice device = mLeDeviceListAdapter.getDevice(position);
        if (device == null) return;
        final Intent intent = new Intent(this, BarGraph.class);
        intent.putExtra(RealTimeGraphing.EXTRAS_DEVICE_NAME,
device.getName());
        intent.putExtra(RealTimeGraphing.EXTRAS_DEVICE_ADDRESS,
device.getAddress());
        intent.putExtra("subject_weight", weight);


        if (mScanning) {
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
            mScanning = false;
        }
        startActivity(intent);
    }
```

```
    //This is the method scanLeDevice that controls how the device scans for BLE
devices
    //If mScanning = true, then start the scan
    //If mScanning = false, then stop the scan
    private void scanLeDevice(final boolean enable) {
        if (enable) {
            // Stops scanning after a pre-defined scan period.
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    mScanning = false;
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);
                    invalidateOptionsMenu();
                }
            }, SCAN_PERIOD);

            mScanning = true;
            mBluetoothAdapter.startLeScan(mLeScanCallback);
        } else {
            mScanning = false;
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
        }
        invalidateOptionsMenu();
    }

    // Adapter for holding devices found through scanning.
    private class LeDeviceListAdapter extends BaseAdapter {
        private ArrayList<BluetoothDevice> mLeDevices;
        private LayoutInflater mInflator;
```

```
public LeDeviceListAdapter() {
    super();
    mLeDevices = new ArrayList<BluetoothDevice>();
    mInflator = MainActivity.this.getLayoutInflater();
}

public void addDevice(BluetoothDevice device) {
    if(!mLeDevices.contains(device)) {
        mLeDevices.add(device);
    }
}

public BluetoothDevice getDevice(int position) {
    return mLeDevices.get(position);
}

public void clear() {
    mLeDevices.clear();
}

@Override
public int getCount() {
    return mLeDevices.size();
}

@Override
public Object getItem(int i) {
    return mLeDevices.get(i);
}
```

```
        @Override
        public long getItemId(int i) {
            return i;
        }

        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {
            ViewHolder viewHolder;
            // General ListView optimization code.
            if (view == null) {
                view = mInflator.inflate(R.layout.listitem_device, null);
                viewHolder = new ViewHolder();
                viewHolder.deviceAddress = (TextView)
view.findViewById(R.id.device_address);
                viewHolder.deviceName = (TextView)
view.findViewById(R.id.device_name);
                view.setTag(viewHolder);
            } else {
                viewHolder = (ViewHolder) view.getTag();
            }

            BluetoothDevice device = mLeDevices.get(i);
            final String deviceName = device.getName();
            if (deviceName != null && deviceName.length() > 0)
                viewHolder.deviceName.setText(deviceName);
            else
                viewHolder.deviceName.setText(R.string.unknown_device);
                viewHolder.deviceAddress.setText(device.getAddress());

            return view;
```

```
        }
    }

    // Device scan callback.
    private BluetoothAdapter.LeScanCallback mLeScanCallback =
            new BluetoothAdapter.LeScanCallback() {


                @Override
                public void onLeScan(final BluetoothDevice device, int rssi, byte[]
scanRecord) {
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            mLeDeviceListAdapter.addDevice(device);
                            mLeDeviceListAdapter.notifyDataSetChanged();
                        }
                    });
                }
            };
    static class ViewHolder {
        TextView deviceName;
        TextView deviceAddress;
    }
}
```

listitem_device.xml

This is the resource file for the MainActivity.java activity. It consists of textviews used to list the device name as well as the device address.

```
<?xml version="1.0" encoding="utf-8"?>


<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="wrap_content">
    <TextView android:id="@+id/device_name"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:textSize="24dp"/>
    <TextView android:id="@+id/device_address"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:textSize="12dp"/>
</LinearLayout>
```

BarGraph.java

The BarGraph.java activity receives data from the Bluetooth connection and graphs data on the current load bar graph and also on the five step history bar graph.

Data is passed through the broadcast receiver first, then put through the getData() method. This method converts data packet from six 8-bit variables into the three original load values by combining the values using the combinebytes() method. The values are summed together and graphed onto the current load bar graph. The getData() method also checks if the step condition has been met. While the step condition equals 1 (load greater than 10% body weight), the local maximum load is tracked. Then the load decreases to less than 10% body weight, the step condition is set to 0 and the local maximum load is graphed on the five step history bar graph.

```
package com.example.app;

import android.app.Activity;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.IBinder;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ExpandableListView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.jjoe64.graphview.BarGraphView;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphViewDataInterface;
import com.jjoe64.graphview.GraphViewSeries;
```

```
import com.jjoe64.graphview.ValueDependentColor;

import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.HashMap;
import java.util.List;
import java.util.UUID;

/**
 * Created by alvinl on 10/6/2014.
 */

public class BarGraph extends Activity {

    private final static String TAG = RealTimeGraphing.class.getSimpleName();

    public static final String EXTRAS_DEVICE_NAME = "DEVICE_NAME";
    public static final String EXTRAS_DEVICE_ADDRESS =
"DEVICE_ADDRESS";
    public String weight = "";

    private static final String UUID_CHAR =      "e7add780-b042-4876-aae1-
112855353dd1";
    private static final UUID UUID_SERVICE =    UUID.fromString("00001234-
0000-1000-8000-00805f9b34fb");
    public static final String okaystring = "uuidCharacteristic";

    public String b = "";
    public String valuea;

    public byte[] value;

    private TextView mConnectionState;
    private TextView mDataField;
    private String mDeviceName;
    private String mDeviceAddress;
    private ExpandableListView mGattServicesList;
    private BluetoothLeService mBluetoothLeService;
    private ArrayList<ArrayList<BluetoothGattCharacteristic>>
mGattCharacteristics =
            new ArrayList<ArrayList<BluetoothGattCharacteristic>>();
```

```
    private boolean mConnected = false;
    private BluetoothGatt mbluetoothgatt;
    private BluetoothGattCharacteristic mNotifyCharacteristic;
    private BluetoothGattCallback mBluetoothCallback;
    private final String LIST_NAME = "NAME";
    private final String LIST_UUID = "UUID";

    public BluetoothGattCharacteristic characteristic;

    private boolean notify = false;

    public final static String ACTION_DATA_AVAILABLE =
            "com.example.bluetooth.le.ACTION_DATA_AVAILABLE";

    private final Handler mHandler = new Handler();
    private Runnable mTimer1;
    private Runnable mTimer2;
    private GraphView heelView;
    private GraphView medialView;
    private GraphView lateralView;
    private BarGraphView totalView;
    private BarGraphView stepView;

    private GraphViewSeries heelSeries;
    private GraphViewSeries medialSeries;
    private GraphViewSeries lateralSeries;
    private GraphViewSeries totalSeries;
    private GraphViewSeries stepSeries;

    public int color;

    public int weightvalue;

    private double graph2LastXValue = 5d;
    private double graph2LastXValue1 = 5d;

    // Code to manage Service lifecycle.
    private final ServiceConnection mServiceConnection = new
ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName componentName,
IBinder service) {
            mBluetoothLeService = ((BluetoothLeService.LocalBinder)
service).getService();
            if (!mBluetoothLeService.initialize()) {
```

```
            Log.e(TAG, "Unable to initialize Bluetooth");
            finish();
        }

        // Automatically connects to the device upon successful start-up
initialization.
        mBluetoothLeService.connect(mDeviceAddress);

    }
    public void onServiceDisconnected(ComponentName componentName) {
        mBluetoothLeService = null;
    }
};

//    Handles various events fired by the Service.
//    ACTION_GATT_CONNECTED: connected to a GATT server.
//    ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
//    ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
//    ACTION_DATA_AVAILABLE: received data from the device. This can be a
result of read
//    or notification operations.
private final BroadcastReceiver mGattUpdateReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
            mConnected = true;
            updateConnectionState(R.string.connected);
            invalidateOptionsMenu();
        } else if
(BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
            mConnected = false;
            updateConnectionState(R.string.disconnected);
            invalidateOptionsMenu();
            clearUI();

        } else if
(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action))
{
            // Show all the supported services and characteristics on the user
interface.

displayGattServices(mBluetoothLeService.getSupportedGattServices());
```

```
        } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action))
{
            String okay = intent.getExtras().getString(BluetoothLeService.okay);
            value = mNotifyCharacteristic.getValue();
            getData(value);

        }
    }
};

// If a given GATT characteristic is selected, check for supported features. This
sample
// demonstrates 'Read' and 'Notify' features. See
// http://d.android.com/reference/android/bluetooth/BluetoothGatt.html for the
complete
// list of supported characteristic features.


private void clearUI() {

}

TextView packetdataText;
TextView ctText;
TextView stepText;
TextView rstepText;
TextView gstepText;
TextView bstepText;
TextView avgstepText;
TextView weightText;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.bargraph);

    final Intent intent = getIntent();
    mDeviceName = intent.getStringExtra(EXTRAS_DEVICE_NAME);
    mDeviceAddress = intent.getStringExtra(EXTRAS_DEVICE_ADDRESS);

    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        weight = extras.getString("subject_weight");
    }

    weightvalue = Integer.parseInt(weight);
```

```
        // Sets up UI references.

        packetdataText =(TextView)findViewById(R.id.packetdata);
        ctText =(TextView)findViewById(R.id.ctforce);
        stepText  =(TextView)findViewById(R.id.steptext);
        rstepText =(TextView)findViewById(R.id.rsteptext);
        gstepText =(TextView)findViewById(R.id.gsteptext);
        bstepText =(TextView)findViewById(R.id.bsteptext);
        avgstepText =(TextView)findViewById(R.id.avgsteptext);
        weightText =(TextView)findViewById(R.id.subjectweighttext);

        weightText.setText(weight);

        getActionBar().setTitle("ATLAS: Real-time Graph");
        getActionBar().setDisplayHomeAsUpEnabled(true);
        Intent gattServiceIntent1 = new Intent(this, BluetoothLeService.class);
        bindService(gattServiceIntent1, mServiceConnection,
BIND_AUTO_CREATE);

        // graph with dynamically generated horizontal and vertical labels

        totalView = new BarGraphView(this, "Total Load (lbs)");  //(context,graph
title)
        stepView = new BarGraphView(this, "Max Load of previous 5 steps");
//        medialView = new LineGraphView(this, "Medial Load (lbs)");
//        lateralView = new LineGraphView(this, "Lateral Load (lbs)");
//        totalView = new LineGraphView(this, "Total Load (lbs)");

        // shaded under the curve
//        ((LineGraphView) heelView).setDrawBackground(true);
//        ((LineGraphView) medialView).setDrawBackground(true);
//        ((LineGraphView) lateralView).setDrawBackground(true);
        //((LineGraphView) totalView).setDrawBackground(true);


//        heelSeries = new GraphViewSeries("Heel curve", new
GraphViewSeries.GraphViewSeriesStyle(Color.rgb(255, 0, 0), 5), new
GraphView.GraphViewData[] {
//                new GraphView.GraphViewData(1, 0d)
//                , new GraphView.GraphViewData(2, 0d)
//                , new GraphView.GraphViewData(3, 0d) // another frequency
//                , new GraphView.GraphViewData(4, 0d)
//                , new GraphView.GraphViewData(5, 0d)
//                , new GraphView.GraphViewData(7, 0d)
//                , new GraphView.GraphViewData(8, 0d)
```

```
//              , new GraphView.GraphViewData(9, 0d)
//              , new GraphView.GraphViewData(10, 0d)
//
//      });
//
//      medialSeries = new GraphViewSeries("Medial curve", new
GraphViewSeries.GraphViewSeriesStyle(Color.rgb(0, 255, 0), 5), new
GraphView.GraphViewData[] {
//              new GraphView.GraphViewData(1, 0d)
//              , new GraphView.GraphViewData(2, 0d)
//              , new GraphView.GraphViewData(3, 0d) // another frequency
//              , new GraphView.GraphViewData(4, 0d)
//              , new GraphView.GraphViewData(5, 0d)
//              , new GraphView.GraphViewData(7, 0d)
//              , new GraphView.GraphViewData(8, 0d)
//              , new GraphView.GraphViewData(9, 0d)
//              , new GraphView.GraphViewData(10, 0d)
//
//      });
//
//      lateralSeries = new GraphViewSeries("Lateral curve", new
GraphViewSeries.GraphViewSeriesStyle(Color.rgb(0, 0, 255), 5), new
GraphView.GraphViewData[] {
//              new GraphView.GraphViewData(1, 0d)
//              , new GraphView.GraphViewData(2, 0d)
//              , new GraphView.GraphViewData(3, 0d) // another frequency
//              , new GraphView.GraphViewData(4, 0d)
//              , new GraphView.GraphViewData(5, 0d)
//              , new GraphView.GraphViewData(7, 0d)
//              , new GraphView.GraphViewData(8, 0d)
//              , new GraphView.GraphViewData(9, 0d)
//              , new GraphView.GraphViewData(10, 0d)
//
//      });

      GraphViewSeries.GraphViewSeriesStyle seriesStyle = new
GraphViewSeries.GraphViewSeriesStyle();
      seriesStyle.setValueDependentColor(new ValueDependentColor() {
         @Override
         public int get(GraphViewDataInterface data) {
            // the higher the more red
            int color = 0;
            if(data.getY()>=30) {
               //color = Color.rgb((int) (150 + ((data.getY() / 250) * 100)), (int) (150 -
((data.getY() / 250) * 100)), (int) (150 - ((data.getY() / 250) * 100)));
               color = Color.rgb(255, 0, 0);
```

```
            }
            else if(data.getY()>=20){
               //color = Color.rgb((int) (150 - ((data.getY() / 250) * 100)),(int) (150 +
((data.getY() / 250) * 100)),  (int) (150 - ((data.getY() / 250) * 100)));
               color = Color.rgb(0, 255, 0);

            }
            else{
               color = Color.rgb((int) (150 - ((data.getY() / 250) * 100)),  (int) (150 -
((data.getY() / 250) * 100)), (int) (150 + ((data.getY() / 250) * 100)));
            }

            return color;
         }
      });

      totalSeries = new GraphViewSeries("Total curve", seriesStyle, new
GraphView.GraphViewData[] {
            new GraphView.GraphViewData(1, 0d)
            , new GraphView.GraphViewData(2, 0d)
            , new GraphView.GraphViewData(3, 0d) // another frequency
            , new GraphView.GraphViewData(4, 0d)
            , new GraphView.GraphViewData(5, 0d)
            , new GraphView.GraphViewData(7, 0d)
            , new GraphView.GraphViewData(8, 0d)
            , new GraphView.GraphViewData(9, 0d)
            , new GraphView.GraphViewData(10, 0d)

      });

      stepSeries = new GraphViewSeries("Step history", seriesStyle, new
GraphView.GraphViewData[] {
            new GraphView.GraphViewData(1, 0d)
            , new GraphView.GraphViewData(2, 0d)
            , new GraphView.GraphViewData(3, 0d) // another frequency
            , new GraphView.GraphViewData(4, 0d)
            , new GraphView.GraphViewData(5, 0d)

      });


//       heelView.addSeries(heelSeries); // data
//       heelView.setViewPort(0, 160);
//       heelView.setManualYAxisBounds(250, 0);
//       heelView.setScrollable(true);
//       heelView.setScalable(true);
```

```
//        heelView.getGraphViewStyle().setNumHorizontalLabels(4);
//        heelView.getGraphViewStyle().setNumVerticalLabels(3);
//        heelView.getGraphViewStyle().setGridColor(Color.GRAY);
//        heelView.setHorizontalLabels(new String[] {"1s", " ", " ", "10s"});
//        heelView.getGraphViewStyle().setTextSize(50);
//
//        medialView.addSeries(medialSeries); // data
//        medialView.setViewPort(0, 160);
//        medialView.setManualYAxisBounds(250,0);
//        medialView.setScrollable(true);
//        medialView.setScalable(true);
//        medialView.getGraphViewStyle().setNumHorizontalLabels(4);
//        medialView.getGraphViewStyle().setNumVerticalLabels(3);
//        medialView.getGraphViewStyle().setGridColor(Color.GRAY);
//        medialView.setHorizontalLabels(new String[] {"1s", " ", " ", "10s"});
//        medialView.getGraphViewStyle().setTextSize(50);
//
//        lateralView.addSeries(lateralSeries); // data
//        lateralView.setViewPort(0, 160);
//        lateralView.setManualYAxisBounds(250, 0);
//        lateralView.setScrollable(true);
//        lateralView.setScalable(true);
//        lateralView.getGraphViewStyle().setNumHorizontalLabels(4);
//        lateralView.getGraphViewStyle().setNumVerticalLabels(3);
//        lateralView.getGraphViewStyle().setGridColor(Color.GRAY);
//        lateralView.setHorizontalLabels(new String[] {"1s", " ", " ", "10s"});
//        lateralView.getGraphViewStyle().setTextSize(50);

        totalView.addSeries(totalSeries); // data
        totalView.setViewPort(0, 0.1);
        totalView.setManualYAxisBounds(100, 0);
        totalView.setScrollable(true);
        //totalView.setScalable(true);
        totalView.getGraphViewStyle().setNumHorizontalLabels(1);
        totalView.getGraphViewStyle().setNumVerticalLabels(3);
        totalView.getGraphViewStyle().setGridColor(Color.GRAY);
        totalView.setHorizontalLabels(new String[] {" "});
        totalView.getGraphViewStyle().setTextSize(50);

        stepView.addSeries(stepSeries); // data
        stepView.setViewPort(0, 3);
        stepView.setManualYAxisBounds(100, 0);
        stepView.setScrollable(true);
        //totalView.setScalable(true);
        stepView.getGraphViewStyle().setNumHorizontalLabels(5);
        stepView.getGraphViewStyle().setNumVerticalLabels(3);
```

```
        stepView.getGraphViewStyle().setGridColor(Color.GRAY);
        stepView.setHorizontalLabels(new String[] {" ", " ", " ", " ", " "});
        stepView.getGraphViewStyle().setTextSize(50);


//        LinearLayout heel_layout = (LinearLayout) findViewById(R.id.heelGraph);
//        heel_layout.addView(heelView);
//
//        LinearLayout medial_layout = (LinearLayout)
findViewById(R.id.medialGraph);
//        medial_layout.addView(medialView);
//
//        LinearLayout lateral_layout = (LinearLayout)
findViewById(R.id.lateralGraph);
//        lateral_layout.addView(lateralView);

        LinearLayout sum_layout = (LinearLayout) findViewById(R.id.totalGraph);
        sum_layout.addView(totalView);

        LinearLayout step_layout = (LinearLayout) findViewById(R.id.stepGraph);
        step_layout.addView(stepView);

    }

    public double c;
    public double d;

    public int i=1;
    public String avgString;

    public String cString;

    public String v1;
    public int v2 = 0;


    public int h1 = 0;
    public int h2 = 0;
    public double ht = 0;

    public int m1 = 0;
    public int m2 = 0;
    public double mt = 0;

    public int l1 = 0;
    public int l2 = 0;
```

```
    public double lt = 0;

    public double tf = 0;
    public double mf = 0;
    public double mf_local = 0;
    public double avgf = 0;
    public double tmf = 0;

    public int num_steps = 0;
    public int num_rsteps = 0;
    public int num_gsteps = 0;
    public int num_bsteps = 0;

    public int top_g_boundary = 80;
    public int bottom_g_boundary = 35;

    public int step_condition = 0;

    public int packetlen;

    public String datastring = "0";

    public void getData(byte[] value) {

        //convert the data packet into a string for display
        v1 = Arrays.toString(value);

        //check packet length
        packetlen = value.length;

        //if all data is present
        if(packetlen == 6) {

            //perform a bitwise and function with 255 to convert into unsigned 8-bit
integer
            h1 = value[0] & 0XFF;
            h2 = value[1] & 0XFF;
            m1 = value[2] & 0XFF;
            m2 = value[3] & 0XFF;
            l1 = value[4] & 0XFF;
            l2 = value[5] & 0XFF;

            //combination of both bytes into one double type number
            ht = combinebytes(h1, h2);
            mt = combinebytes(m1, m2);
            lt = combinebytes(l1, l2);
```

```
        //summation of all three sensor values and conversion to percentage
        tf = ((ht + mt + lt)/weightvalue)*100;

        //reset back to zero for next reading
        h1 = 0;
        h2 = 0;
        m1 = 0;
        m2 = 0;
        l1 = 0;
        l2 = 0;

        avgString = Double.toString(tf);
        cString = Double.toString(mt);

//          if(ht>100){
//              heelView.setBackgroundColor(Color.RED);
//          }else if(ht>50){
//              heelView.setBackgroundColor(Color.GREEN);
//          } else{
//              heelView.setBackgroundColor(Color.YELLOW);
//          }
//
//          if(mt>100){
//              medialView.setBackgroundColor(Color.RED);
//          }else if(mt>50){
//              medialView.setBackgroundColor(Color.GREEN);
//          } else{
//              medialView.setBackgroundColor(Color.YELLOW);
//          }
//
//          if(lt>100){
//              lateralView.setBackgroundColor(Color.RED);
//          }else if(lt>50){
//              lateralView.setBackgroundColor(Color.GREEN);
//          } else{
//              lateralView.setBackgroundColor(Color.YELLOW);
//          }
//
//          if(tf>100){
//              totalView.setBackgroundColor(Color.RED);
//          }else if(tf>50){
//              totalView.setBackgroundColor(Color.GREEN);
//          } else{
//              totalView.setBackgroundColor(Color.YELLOW);
//          }
```

```
        //Finding local maximum
        //If total force is greater than 10
        if(tf > mf_local){
            mf_local = tf;

            if(tf>10){
                step_condition = 1;
            }

            if(mf_local > mf){
            mf = mf_local;
            }
        }

        if(step_condition == 1) {          ///////Step condition is met when a step is
greater than 10% body weight and the end step condition is met when the step is
less than 15 lbs
            if(tf<10) {
                graph2LastXValue1 += 1d;
                stepSeries.appendData(new
GraphView.GraphViewData(graph2LastXValue1, mf), true, 1600);

                if (mf >= top_g_boundary) {
                    num_rsteps++;
                } else if (mf >= bottom_g_boundary) {
                    num_gsteps++;
                } else {
                    num_bsteps++;
                }

                num_steps++;

                tmf = tmf + mf;

                datastring = datastring+","+mf;

                avgf = tmf/num_steps;

                //Updating text views Red = over threshold steps Green = Within
threshold Gray = under threshold
                stepText.setText("Steps: " + num_steps);
                rstepText.setText("Red Steps: " + num_rsteps);
                gstepText.setText("Green Steps: " + num_gsteps);
                bstepText.setText("Gray Steps: " + num_bsteps);
```

```
                //Average max load
                avgstepText.setText(String.format("Average max load: %.2f", avgf)
+"%");

                mf_local = 0;
                mf = 0;
                step_condition = 0;
            }
        }

        ctText.setText(String.format("Current load: %.2f",tf) +"%");
        packetdataText.setText("Packet data: " + v1);


        //iterate graph index and graph most recent total force value
        graph2LastXValue += 1d;
//          heelSeries.appendData(new
GraphView.GraphViewData(graph2LastXValue, ht), true, 1600);
//          medialSeries.appendData(new
GraphView.GraphViewData(graph2LastXValue, mt), true, 1600);
//          lateralSeries.appendData(new
GraphView.GraphViewData(graph2LastXValue, lt), true, 1600);
        totalSeries.appendData(new
GraphView.GraphViewData(graph2LastXValue, tf), true, 1600);
        graph2LastXValue += 1d;
        totalSeries.appendData(new
GraphView.GraphViewData(graph2LastXValue, tf), true, 1600);
        i++;
      }
   }


   public double combinebytes(int v0, int v3) {

      if (v3 == 0) {
         v2 = v0;
      } else if (v3 == 1) {
         v2 = v0 + 256;
      } else if (v3 == 2) {
         v2 = v0 + 512;
      } else if (v3 == 3) {
         v2 = v0 + 768;
      } else if (v3 == 4) {
         v2 = v0 + 1024;
      } else if (v3 == 5) {
```

```
    v2 = v0 + 1280;
} else if (v3 == 6) {
    v2 = v0 + 1536;
} else if (v3 == 7) {
    v2 = v0 + 1792;
} else if (v3 == 8) {
    v2 = v0 + 2048;
} else if (v3 == 9) {
    v2 = v0 + 2304;
} else if (v3 == 10) {
    v2 = v0 + 2560;
} else if (v3 == 11) {
    v2 = v0 + 2816;
} else if (v3 == 12) {
    v2 = v0 + 3072;
} else if (v3 == 13) {
    v2 = v0 + 3328;
} else if (v3 == 14) {
    v2 = v0 + 3584;
} else if (v3 == 15) {
    v2 = v0 + 3840;
} else if (v3 == 16) {
    v2 = v0 + 4096;
} else if (v3 == 17) {
    v2 = v0 + 4352;
} else if (v3 == 18) {
    v2 = v0 + 4608;
} else if (v3 == 19) {
    v2 = v0 + 4864;
} else if (v3 == 20) {
    v2 = v0 + 5120;
} else if (v3 == 21) {
    v2 = v0 + 5376;
} else if (v3 == 22) {
    v2 = v0 + 5632;
} else if (v3 == 23) {
    v2 = v0 + 5888;
} else if (v3 == 24) {
    v2 = v0 + 6144;
} else if (v3 == 25) {
    v2 = v0 + 6400;
} else if (v3 == 26) {
    v2 = v0 + 6656;
} else if (v3 == 27) {
    v2 = v0 + 6912;
} else if (v3 == 28) {
```

```
    v2 = v0 + 7168;
} else if (v3 == 29) {
    v2 = v0 + 7424;
} else if (v3 == 30) {
    v2 = v0 + 7680;
} else if (v3 == 31) {
    v2 = v0 + 7936;
} else if (v3 == 32) {
    v2 = v0 + 8192;
} else if (v3 == 33) {
    v2 = v0 + 8448;
} else if (v3 == 34) {
    v2 = v0 + 8704;
} else if (v3 == 35) {
    v2 = v0 + 8960;
} else if (v3 == 36) {
    v2 = v0 + 9216;
} else if (v3 == 37) {
    v2 = v0 + 9472;
} else if (v3 == 38) {
    v2 = v0 + 9728;
} else if (v3 == 39) {
    v2 = v0 + 9984;
} else if (v3 == 40) {
    v2 = v0 + 10240;
} else if (v3 == 41) {
    v2 = v0 + 10496;
} else if (v3 == 42) {
    v2 = v0 + 10752;
} else if (v3 == 43) {
    v2 = v0 + 11008;
} else if (v3 == 44) {
    v2 = v0 + 11264;
} else if (v3 == 45) {
    v2 = v0 + 11520;
} else if (v3 == 46) {
    v2 = v0 + 11776;
} else if (v3 == 47) {
    v2 = v0 + 12032;
} else if (v3 == 48) {
    v2 = v0 + 12288;
} else if (v3 == 49) {
    v2 = v0 + 12544;
} else if (v3 == 50) {
    v2 = v0 + 12800;
} else if (v3 == 51) {
```

```
        v2 = v0 + 13056;
    } else if (v3 == 52) {
        v2 = v0 + 13312;
    } else if (v3 == 53) {
        v2 = v0 + 13568;
    } else if (v3 == 54) {
        v2 = v0 + 13824;
    } else if (v3 == 55) {
        v2 = v0 + 14080;
    } else if (v3 == 56) {
        v2 = v0 + 14336;
    } else if (v3 == 57) {
        v2 = v0 + 14592;
    } else if (v3 == 58) {
        v2 = v0 + 14848;
    } else if (v3 == 59) {
        v2 = v0 + 15104;
    } else if (v3 == 60) {
        v2 = v0 + 15360;
    } else if (v3 == 61) {
        v2 = v0 + 15616;
    } else if (v3 == 62) {
        v2 = v0 + 15872;
    } else if (v3 == 63) {
        v2 = v0 + 16128;
    }

    if(v0+v3==0) {
        v2 = 0;
    }

    c = (1.0*(v2));

    //d = c*3.3/4096;

    d = c*250/16384;

    return d;
}


@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter());
    if (mBluetoothLeService != null) {
```

```
            final boolean result = mBluetoothLeService.connect(mDeviceAddress);
            Log.d(TAG, "Connect request result=" + result);
        }

    }


    FileOutputStream outputStream;
    String filename = "myfile";
    String time = "";

    //on Pause, a save file is created in the folder Verification Data

    @Override
    protected void onPause() {
        super.onPause();
        mHandler.removeCallbacks(mTimer1);
        unregisterReceiver(mGattUpdateReceiver);


        packetdataText.setText("--");
        ctText.setText("--");


        Calendar c = Calendar.getInstance();
        int month = c.get(Calendar.MONTH) + 1;
        int day = c.get(Calendar.DAY_OF_MONTH);
        int hour24 = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);
        int seconds = c.get(Calendar.SECOND);


        //Format Mo, DD, HH, MM, Duration
        filename = "Atlas "+ weight + " " + String.format("%02d, %02d, %02d,
%02d", month, day, hour24, minute);


        time = hour24+":"+minute;


        generateTextfile(filename,datastring);
        Toast.makeText(this, "Saved as " + time, Toast.LENGTH_SHORT).show();
    }

    //Creates a new file in the Verification data file. Title = sFilename. Content =
sBody
```

```
    public void generateTextfile(String sFileName, String sBody){
        try
        {
            File root = new File(Environment.getExternalStorageDirectory(),
"Verification Data");
            if (!root.exists()) {
                root.mkdirs();
            }
            File textfile = new File(root, sFileName);
            FileWriter writer = new FileWriter(textfile);
            writer.append(sBody);
            writer.flush();
            writer.close();
            Toast.makeText(this, "Saved", Toast.LENGTH_SHORT).show();
        }
        catch(IOException e)
        {
            e.printStackTrace();

        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        //unbindService(mServiceConnection);
        //mBluetoothLeService = null;
        Toast.makeText(this, "here2", Toast.LENGTH_SHORT).show();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.gatt_services, menu);
        if (mConnected) {
            menu.findItem(R.id.menu_connect).setVisible(false);
            menu.findItem(R.id.menu_disconnect).setVisible(true);
        } else {
            menu.findItem(R.id.menu_connect).setVisible(true);
            menu.findItem(R.id.menu_disconnect).setVisible(false);
        }
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()) {
```

```
            case R.id.menu_connect:
                mBluetoothLeService.connect(mDeviceAddress);
                return true;
            case R.id.menu_disconnect:
                mBluetoothLeService.disconnect();
                return true;
            case android.R.id.home:
                onBackPressed();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    private void updateConnectionState(final int resourceId) {
        runOnUiThread( new Runnable() {
            @Override
            public void run() {
            }
        });
    }

    private void displayGattServices(List<BluetoothGattService> gattServices) {
        if (gattServices == null) return;
        String uuid = null;
        String unknownServiceString =
getResources().getString(R.string.unknown_service);
        String unknownCharaString =
getResources().getString(R.string.unknown_characteristic);
        ArrayList<HashMap<String, String>> gattServiceData = new
ArrayList<HashMap<String, String>>();
        ArrayList<ArrayList<HashMap<String, String>>> gattCharacteristicData
                = new ArrayList<ArrayList<HashMap<String, String>>>();
        mGattCharacteristics = new
ArrayList<ArrayList<BluetoothGattCharacteristic>>();

        // Loops through available GATT Services.
        for (BluetoothGattService gattService : gattServices) {
            HashMap<String, String> currentServiceData = new HashMap<String,
String>();
            uuid = gattService.getUuid().toString();
            currentServiceData.put(
                    LIST_NAME, SampleGattAttributes.lookup(uuid,
unknownServiceString));
            currentServiceData.put(LIST_UUID, uuid);
            gattServiceData.add(currentServiceData);
```

```
        ArrayList<HashMap<String, String>> gattCharacteristicGroupData =
                new ArrayList<HashMap<String, String>>();
        List<BluetoothGattCharacteristic> gattCharacteristics =
                gattService.getCharacteristics();
        ArrayList<BluetoothGattCharacteristic> charas =
                new ArrayList<BluetoothGattCharacteristic>();

        // Loops through available Characteristics.
        for (BluetoothGattCharacteristic gattCharacteristic : gattCharacteristics) {
            charas.add(gattCharacteristic);
            HashMap<String, String> currentCharaData = new HashMap<String,
String>();
            uuid = gattCharacteristic.getUuid().toString();
            characteristic = gattCharacteristic;
            currentCharaData.put(
                    LIST_NAME, SampleGattAttributes.lookup(uuid,
unknownCharaString));
            currentCharaData.put(LIST_UUID, uuid);
            gattCharacteristicGroupData.add(currentCharaData);
        }
        mGattCharacteristics.add(charas);
        gattCharacteristicData.add(gattCharacteristicGroupData);
      }

      mNotifyCharacteristic = characteristic;

      String UUID = characteristic.getUuid().toString();
      Toast.makeText(BarGraph.this, UUID, Toast.LENGTH_SHORT).show();
      mBluetoothLeService.setCharacteristicNotification(
            characteristic, true);
   }

   private static IntentFilter makeGattUpdateIntentFilter() {
      final IntentFilter intentFilter = new IntentFilter();
      intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);
      intentFilter.addAction(
BluetoothLeService.ACTION_GATT_DISCONNECTED);

intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOV
ERED);
      intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE);
      return intentFilter;
```

bargraph.xml

This is the resource file for the BarGraph.java activity. It consists of the current load graph, five step history graph, as well as textviews to display information such as total number of steps to the user.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout
        android:id="@+id/totalGraph"
        android:layout_width="fill_parent"
        android:layout_height="800dp"
        android:orientation="vertical">

    </LinearLayout>

    <LinearLayout
        android:id="@+id/stepGraph"
        android:layout_width="fill_parent"
        android:layout_height="250dp"
        android:layout_below="@+id/totalGraph"
        android:orientation="vertical">

    </LinearLayout>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="--"
        android:id="@+id/packetdata"
        android:layout_alignTop="@+id/ctforce"
        android:layout_toRightOf="@+id/subjectweighttext"
        android:layout_marginLeft="20dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="--"
        android:id="@+id/ctforce"
        android:layout_below="@+id/stepGraph"
        android:layout_alignParentStart="true"
        android:layout_marginTop="20dp" />

    <TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="--"
    android:id="@+id/avgsteptext"
    android:layout_alignTop="@+id/ctforce"
    android:layout_toRightOf="@+id/ctforce"
    android:layout_marginLeft="20dp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="--"
    android:id="@+id/subjectweighttext"
    android:layout_alignTop="@+id/ctforce"
    android:layout_toRightOf="@+id/avgsteptext"
    android:layout_marginLeft="20dp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="--"
    android:id="@+id/steptext"
    android:layout_below="@+id/packetdata"
    android:layout_alignParentStart="true"
    android:layout_marginTop="20dp"
    android:textSize="80dp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="--"
    android:id="@+id/bsteptext"
    android:layout_alignTop="@+id/steptext"
    android:layout_toRightOf="@+id/steptext"
    android:layout_marginLeft="20dp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="--"
    android:id="@+id/gsteptext"
    android:layout_alignTop="@+id/steptext"
    android:layout_toRightOf="@+id/bsteptext"
    android:layout_marginLeft="20dp" />

<TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="--"
        android:id="@+id/rsteptext"
        android:layout_alignTop="@+id/steptext"
        android:layout_toRightOf="@+id/gsteptext"
        android:layout_marginLeft="20dp" />


</RelativeLayout>
```

REFERENCES

1. Buckwalter, Joseph A., and Alan J. Grodzinsky. "Loading of healing bone, fibrous tissue, and muscle: implications for orthopaedic practice." *Journal of the American Academy of Orthopaedic Surgeons* 7.5 (1999): 291-299.

2. Rubin, Clinton, et al. "Differentiation of the Bone-Tissue Remodeling Response to Axial and Torsional Loading in the Turkey Ulna*†." *The Journal of Bone & Joint Surgery* 78.10 (1996): 1523-33.

3. Goodship, A. E., L. E. Lanyon, and H. McFie. "Functional adaptation of bone to increased stress. An experimental study." *The Journal of Bone & Joint Surgery* 61.4 (1979): 539-546.

4. Anahad O'Connor (October 18, 2010). "The Claim: After Being Broken, Bones Can Become Even Stronger". New York Times.

5. Frost, HM (1994). "Wolff's Law and bone's structural adaptations to mechanical usage: an overview for clinicians". *The Angle Orthodontist* 64 (3): 175–188. doi:10.1043/0003-3219(1994)064<0175:WLABSA>2.0.CO;2. PMID 8060014.

6. Ruff, Christopher; Holt, Brigitte; Trinkaus, Erik (April 2006). "Who's afraid of the big bad Wolff?: "Wolff's law" and bone functional adaptation". *American Journal of Physical Anthropology* 129 (4): 484–498. doi:10.1002/ajpa.20371. Retrieved 2 March 2015.

7. Sarmiento, AUGUSTO, et al. "Fracture healing in rat femora as affected by functional weight-bearing." *The Journal of Bone & Joint Surgery* 59.3 (1977): 369-375.

8. Meadows, TIMOTHY H., et al. "Effect of weight-bearing on healing of cortical defects in the canine tibia." *The Journal of Bone & Joint Surgery* 72.7 (1990): 1074-1080.

9. Budsberg, Steven C., Mary C. Verstraete, and Robert W. Soutas-Little. "Force plate analysis of the walking gait in healthy dogs." *American Journal of Veterinary Research* 48.6 (1987): 915-918.

10. Vasarhelyi, Attila, et al. "Partial weight bearing after surgery for fractures of the lower extremity–is it achievable?" *Gait & Posture* 23.1 (2006): 99-105.

11. Hustedt, Joshua W., et al. "Is it possible to train patients to limit weight bearing on a lower extremity." *Orthopedics* 35.1 (2012): e31-e37.

12. Hershko, Erel, Chanan Tauber, and Eli Carmeli. "Biofeedback versus physiotherapy in patients with partial weight-bearing." *American Journal of Orthopedics* 37.5 (2008): E92-E96.

13. Pataky, Zoltan, et al. "Biofeedback training for partial weight bearing in patients after total hip arthroplasty." *Archives of Physical Medicine and Rehabilitation* 90.8 (2009): 1435-1438.

14. Winstein, Carolee J., et al. "Learning a partial-weight-bearing skill: effectiveness of two forms of feedback." *Physical Therapy* 76.9 (1996): 985-993.

15. Warren CG, Lehmann JF. Training procedures and biofeedback methods to achieve controlled partial weight bearing: an assessment. *Archives of Physical Medicine and Rehabilitation*. 1975; 56(10):449-455.